BEST AVAILABLE COPY

10/664,740

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number : **0 569 133 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **93302717.9**

(51) Int. Cl.[5] : **G06F 15/20**, G06F 9/44

(22) Date of filing : **07.04.93**

(30) Priority : **08.04.92 US 866658**

(43) Date of publication of application :
10.11.93 Bulletin 93/45

(84) Designated Contracting States :
AT BE CH DE DK ES FR GB GR IE IT LI LU MC NL PT SE

(71) Applicant : **BORLAND INTERNATIONAL, Inc.**
1800 Green Hills Road
Scott's Valley, California 95067 (US)

(72) Inventor : **Anderson, Charles R.**
323 Arroyo Seco
Santa Cruz, California 95060 (US)

Inventor : **Warfield, Robert W.**
3470 Merrill Street
Aptos, California 95003 (US)
Inventor : **Cserl, Istvan**
15 Terrace View Court
Scotts Valley, California 95066 (US)
Inventor : **Low, Murray K.**
802 Columbia Street
Santa Cruz, California 95060 (US)
Inventor : **Liaw, Weikuo**
104 Nanna Court
Santa Cruz, California 95060 (US)
Inventor : **Bush, Alan M.**
905 Fremont Place, No.4
Menlo Park, California 94025 (US)

(74) Representative : **Godsill, John Kenneth et al**
Haseltine Lake & Co. Hazlitt House 28 Southampton Buildings Chancery Lane
London WC2A 1AT (GB)

(54) System and methods for computer interfaces.

(57) An electronic interface to a spreadsheet system includes a notebook interface (200) having a plurality of notebook pages (250), each of which may contain a spread of information cells, or other desired page type (e.g., Graphs page 700). Methods (900, 920, 940, 1050) are provided for rapidly accessing and processing information on the different pages, including displaying a plurality of page identifiers for selecting individual pages, and further including a preferred syntax for referencing information. Additional methods are provided for editing cells and blocks of cells.

EP 0 569 133 A2

BNSDOCID: <EP___0569133A2_I_>

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

The present invention relates generally to the field of information processing by digital computers and, more particularly, to the interfacing with the processing and presentation of information by program applications, particularly electronic spreadsheets.

Before computers, numerical analyses, particularly financial ones, were usually prepared on an accountant's columnar pad or spreadsheet, with pencil and calculator in hand. By organizing data into columns and rows, spreadsheets afford the rapid assimilation of information by a reader. The task of preparing a spreadsheet on paper, however, is not quite so fast. Instead, the process tends to be very slow, as each entry must be tediously calculated and entered into the spreadsheet. Since all calculations are the responsibility of the preparer, manually prepared spreadsheets are also prone to errors. Hence, preparation of spreadsheets by hand is slow, tedious, and unreliable.

With the advent of microcomputers, a solution was forthcoming in the form of "electronic spreadsheets." Better known simply as "spreadsheets," these software programs provide a computerized replacement for the traditional financial modeling tools: the accountant's columnar pad, pencil, and calculator. In some regards, spreadsheet programs are to those tools what wordprocessors are to typewriters. Spreadsheets offer dramatic improvements in ease of creating, editing, and using financial models.

A typical spreadsheet program configures the memory of a computer to resemble the column/row or grid format of an accountant's columnar pad, thus providing a visible calculator for a user. Because this "pad" exists dynamically in the computer's memory, however, it differs from paper pads in several important ways. Locations in the electronic spreadsheet, for example, must be communicated to the computer in a format which it can understand. A common scheme for accomplishing this is to assign a number to each row in a spreadsheet, and a letter to each column. To reference a location at column A and row 1 (i.e., the upper-lefthand corner), for example, the user types in "A1". In this manner, the spreadsheet defines an addressable storage location or "cell" at each intersection of a row with a column.

Data entry into an electronic spreadsheet occurs in much the same manner that information would be entered on an accountant's pad. After a screen cursor is positioned at a desired location, the user can enter alphanumeric information. Besides holding text and numeric information, however, spreadsheet cells can store special instructions or "formulas" specifying calculations to be performed on the numbers stored in spreadsheet cells. In this fashion, cell references can serve as variables in an equation, thereby allowing precise mathematical relationships to be defined between cells. The structure and operation of a spreadsheet program, including advanced functions such as functions and macros, are documented in the technical, trade, and patent literature. For an overview, see e.g., Cobb, S., *Using Quattro Pro 2*, Borland-Osborne/McGraw-Hill, 1990; and LeBlond, G. and Cobb, D., *Using 1-2-3*, Que Corp., 1985.

Electronic spreadsheets offer many advantages over their paper counterparts. For one, electronic spreadsheets are much larger (i.e., hold more information) than their paper counterparts; electronic spreadsheets having thousands or even millions of cells are not uncommon. Spreadsheet programs also allow users to perform "what if" scenarios. After a set of mathematical relationships has been entered into a worksheet, the spread of information can be recalculated using different sets of assumptions, with the results of each recalculation appearing almost instantaneously. Performing this operation manually, with paper and pencil, would require recalculating every relationship in the model with each change made.

While electronic spreadsheets offer significant productivity gains in the task of complex data modeling, none has been as intuitive to use as ordinary paper and pencil -- objects already familiar to the user. Instead, the user must master many complex and arbitrary operations. To find the proper command for a task at hand, for example, the user must hunt through a complex menuing system, with the desired choice often buried under several menus. Even simple tasks can pose a significant challenge to the user. To change the punctuation format of a number in one prior art spreadsheet, for example, the user must traverse several nodes of a menu tree, carefully selecting among cryptic menu choices along the way. A mistake at any one of the nodes can lead to harsh consequences, including the loss of valuable data.

Finding this approach to be unworkable, many users memorize frequently-needed commands instead. To accomplish the foregoing task, for example, the user would memorize the command: /Worksheet Global Default Other International. As one can only memorize just so many arbitrary commands, however, the user typically masters only a very small subset of available commands and features. And without constant practice, these commands tend to be quickly forgotten. Moreover, many useful and needed commands are sufficiently hidden in layers of menus that they are never discovered by the user. All told, the non-intuitive interfaces of prior art spreadsheets have led to steep learning curves for users. Even after mastering a particular spread-

BNSDOCID: <EP__0569133A2_I_>

sheet interface, the user typically only knows a fraction of available commands and features, most of which are easily forgotten.

Even with advances in computer and software technology, electronic spreadsheets have not necessarily become easier to use. Instead, technological advances have been largely employed to build more complex functions and modeling features into spreadsheets, often with more complicated menu trees; or worse yet, a staggering array of icons which leave the user even more bewildered. Thus, while prior art spreadsheets have continued to increase in functionality, they have also greatly increased in complexity for the user.

Three dimensionality is one such example. Three-dimensional spreadsheets allow the user to create a worksheet having cells arranged in a 3-D grid. In this manner, the user can manipulate multi-dimensional ranges, i.e., solid blocks of cells. This feature has distinct advantages. For example, the user can build a worksheet consisting of multiple two-dimensional spreads, define 3-D ranges that span these spreads, and copy a range of rows and columns into each of many 2-D spreads at once. This feature eases difficult chores such as consolidation of multiple spreads.

Despite its advantages, three-dimensionality, as presently implemented, is an advanced feature beyond the grasp of many spreadsheet users. This is not a necessary result of a three-dimensional model per se but, instead, has resulted from poor implementations of the model in prior art spreadsheet programs. One popular implementation of the model in the prior art, for example, requires the user to manipulate each additional spread of a three-dimensional spreadsheet as a separate window in a graphical windowing environment. This approach is far from intuitive, however. In particular, this approach requires the user to master actions which have no counterpart in everyday activities. While three-dimensional spreadsheets provide additional functionality, they serve to illustrate how non-intuitive implementations of new technology greatly add to the complexity of the user interface.

Various aspects of the invention are exemplified by the appended claims. From these claims and the following description, it will be appreciated that one can develop a highly intuitive interface.

More particularly, one may provide interface objects which are familiar to the user. In this manner, the user will not have to master an elaborate and/or awkward environment but, instead, may rely upon his or her own common fund of knowledge. Embodiments of the present invention can fulfil this and other needs. In an exemplary embodiment, an electronic spreadsheet system includes a notebook interface having a plurality of notebook pages, each of which contains a spread of information cells, or other desired page type (e.g., Graphs page).

Methods can be provided for rapidly accessing and processing information on the different pages, including, for example, displaying a plurality of page identifiers for selecting individual pages. Additional methods can be provided for editing cells and blocks of cells. In this fashion, a spreadsheet notebook can be designed which provides a convenient interface for organizing and presenting information. Moreover, such a spreadsheet notebook can readily accommodate complex data (e.g., consolidation across multiple spreads) yet, at the same time, can provide the user with a highly intuitive interface, one which employs interface objects which are familiar to the user.

One may also include system and methods for conveniently inspecting and setting the properties of objects. One method for accessing an object's property, includes receiving a request from the user for inspection of an object; accessing properties for the object; and displaying the properties to the user. From the displayed properties, the user may alter the characteristics of the object, as desired.

For a better understanding of the invention and to show how the same may be carried into effect, reference will now be made, by way of example, to the accompanying drawings, wherein:

Fig. 1A is a block diagram of a computer system in which the present invention may be embodied.

Fig. 1B is a block diagram of a software system of the present invention, which includes operating system, application software, and user interface components.

Fig. 1C is a bitmap screen shot illustrating the basic architecture and functionality of a graphical user interface of an embodiment.

Fig. 2A is a screen bitmap illustrating a spreadsheet notebook of an embodiment.

Fig. 2B is a bitmap of a tool bar component of the spreadsheet of the embodiment.

Fig. 2C is a screen bitmap illustrating a notebook window from the spreadsheet notebook of the embodiment.

Figs. 2D-E are bitmaps illustrating page identifiers for rapidly accessing and manipulating individual pages of the notebook.

Figs. 3A-C illustrate navigation (i.e. access of desired information) in the spreadsheet notebook.

Figs. 4A-E are screen bitmaps illustrating the selection of information cells in the spreadsheet notebook.

Fig. 4F is a screen bitmap illustrating the operation of grouping one or more desired pages in the spreadsheet notebook.

BNSDOCID: <EP__0569133A2_I_>

Fig. 4G is a screen bitmap illustrating drag-and-drop editing in the spreadsheet notebook.

Figs. 4H-J are a series of screen bitmaps illustrating a model copy method.

Fig. 4K is a set of bitmaps illustrating the operation of moving and copying a notebook page.

Fig. 4L-M are screen bitmaps illustrating the presentation of information in a notebook.

Fig. 5A is a screen bitmap illustrating exemplary objects of the spreadsheet notebook.

Fig. 5B-E are bitmaps illustrating property inspectors for the objects of Fig. 5A.

Figs. 6A-K are bitmaps illustrating different states (and accompanying panes) for the property inspector of Fig. 5E, each state depending on a particular property of the object being inspected.

Fig. 7A is a screen bitmap illustrating a graph window, with different graph objects available for property inspection.

Fig. 7B-H are bitmaps illustrating exemplary property inspectors for the graph objects of Fig. 7A.

Fig. 8A is a block diagram illustrating the structure and operation of a spreadsheet notebook.

Figs. 8B-C illustrate the correspondence between a page table data structure and pages which are displayed on the screen to the user.

Figs. 8D-F illustrate the referencing of information in the spreadsheet notebook.

Fig. 9A is a flowchart illustrating a method for interpreting information references.

Fig. 9B is a flowchart illustrating a method for drag and drop editing.

Fig. 9C is a flowchart illustrating a method for model copying.

Fig. 10A is a block diagram illustrating a system class hierarchy which is employed for property inspection.

Fig. 10B is a block diagram illustrating the correspondence between a parent object and a child object, and their respective property lists.

Fig. 10C is a flowchart illustrating a method for inspecting and setting properties of objects.

Fig. 10D is a flowchart illustrating a set property method (substep from the method of Fig. 10C).

System Hardware

As shown in Fig. 1A, the present invention may be embodied in a computer system such as the system 100, which comprises a central processor 101, a main memory 102, an input/output controller 103, a keyboard 104, a pointing device 105 (e.g., mouse, track ball, pen device, or the like), a display device 106, and a mass storage 107 (e.g., hard disk). Additional input/output devices, such as a printing device 108, may be included in the system 100 as desired. As illustrated, the various components of the system 100 communicate through a system bus 110 or similar architecture. In a preferred embodiment, the computer system 100 includes an IBM-compatible personal computer, which is available from several vendors (including IBM of Armonk, NY).

Illustrated in Fig. 1B, a computer software system 150 is provided for directing the operation of the computer system 100. Software system 150, which is stored in system memory 102 and on disk memory 107, includes a kernel or operating system 151 and a shell or interface 153. One or more application programs, such as application software 152, may be "loaded" (i.e., transferred from storage 107 into memory 102) for execution by the system 100. The system 100 receives user commands and data through user interface 153; these inputs may then be acted upon by the system 100 in accordance with instructions from operating module 151 and/or application module 152. The interface 153, which is preferably a graphical user interface (GUI), also serves to display results, whereupon the user may supply additional inputs or terminate the session. In a preferred embodiment, operating system 151 is MS-DOS, and interface 153 is Microsoft® Windows; both are available from Microsoft Corporation of Redmond, Washington. Application module 152, on the other hand, includes a spreadsheet notebook (as described in further detail hereinafter).

Interface: User-familiar Objects

A. Introduction

The following description will focus on the presently preferred embodiments of the present invention, which are embodied in spreadsheet applications operative in the Microsoft Windows environment. The present invention, however, is not limited to any particular application or any particular environment. Instead, those skilled in the art will find that the system and methods of the present invention may be advantageously applied to a variety of system and application software, including database management systems, wordprocessors, and the like. Moreover, the present invention may be embodied on a variety of different platforms, including Macintosh, UNIX, NextStep, and the like. Therefore, the description of the exemplary embodiments which follows is for purposes of illustration and not limitation.

Referring now to Fig. 1C, the system 100 includes a windowing interface or workspace 160. Window 160

BNSDOCID: <EP___0569133A2_I_>

is a rectangular, graphical user interface (GUI) for display on screen 106; additional windowing elements may be displayed in various sizes and formats (e.g., tiled or cascaded), as desired. At the top of window 160 is a menu bar 170 with a plurality of user-command choices, each of which may invoke additional submenus and software tools for use with application objects. Window 160 includes a client area 180 for displaying and manipulating screen objects, such as graphic object 181 and text object 182. In essence, the client area is a workspace or viewport for the user to interact with data objects which reside within the computer system 100.

Windowing interface 160 includes a screen cursor or pointer 185 for selecting and otherwise invoking screen objects of interest. In response to user movement signals from the pointing device 105, the cursor 185 floats (i.e., freely moves) across the screen 106 to a desired screen location. During or after cursor movement, the user may generate user-event signals (e.g., mouse button "clicks" and "drags") for selecting and manipulating objects, as is known in the art. For example, Window 160 may be closed, resized, or scrolled by "clicking" (selecting) screen components 172, 174/5, and 177/8, respectively.

In a preferred embodiment, screen cursor 185 is controlled with a mouse device. Single-button, double-button, or triple-button mouse devices are available from a variety of vendors, including Apple Computers of Cupertino, CA, Microsoft Corporation of Redmond, WA, and Logitech Corporation of Fremont, CA, respectively. More preferably, screen cursor control device 105 is a two-button mouse device, including both right and left "mouse buttons." Programming techniques and operations for mouse devices are well documented in the programming and hardware literature; see e.g., *Microsoft Mouse Programmer's Reference*, Microsoft Press, 1989.

B. Spreadsheet Notebooks

In contrast to conventional applications, even those operative in a windowing environment, the present embodiment includes user-familiar objects, i.e., paradigms of real-world objects which the user already knows how to use. In an exemplary embodiment, system 100 includes a spreadsheet notebook interface, with such user-familiar objects as pages and tabs. In this manner, complexities of the system are hidden under ordinary, everyday object metaphors. By drawing upon skills which the user has already mastered, the present invention provides a highly intuitive interface -- one in which advanced features (e.g., three-dimensionality) are easily learned.

1. General Interface

Shown in Fig. 2A, a spreadsheet notebook interface of the present embodiment will now be described. The spreadsheet notebook or workbook of the present embodiment includes a notebook workspace 200 for receiving, processing, and presenting information, including alphanumeric as well as graphic information. Notebook workspace 200 includes a menu bar 210, a tool bar 220, a current cell indicator 230, an input line 231, a status line 240, and a notebook window 250. The menu bar 210 displays and invokes, in response to user inputs, a main level of user commands. Menu 210 also invokes additional pulldown menus, as is known in windowing applications. Input line 231 accepts user commands and information for the entry and editing of cell contents, which may include data, formulas, macros, and the like. Indicator 230 displays an address for the current cursor (i.e., active cell) position. At the status line 240, system 100 displays information about the current state of the workbook; for example, a "READY" indicator means that the system is ready for the user to select another task to be performed.

The tool bar 220, shown in further detail in Fig. 2B, comprises a row or palette of tools which provide a quick way for the user to choose commonly-used menu commands or properties. In an exemplary embodiment, tool bar 220 includes cut, copy, and paste buttons 221, a power button tool 222, a graph tool 223, alignment buttons 224, a style list 225, font buttons 226, insert/delete buttons 227, a fit button 228, and action buttons 229. Buttons 221 cut, copy and paste data and objects to and from Windows clipboard. The same actions are also available as corresponding commands in the Edit menu (available from menu bar 210). Tool 220 creates "powerbuttons" which allow a user to run spreadsheet macros; in a specific embodiment, powerbuttons appear as floating objects in a layer above spreadsheet cells. In a similar fashion, the graph tool 223 creates floating graphs that appear above spreadsheet cells.

Other tools are available for formatting cells. Alignment buttons 224 place cell entries flush left, centered, or flush right, as desired. The style list 225 specifies the style for the active block and may be selected from a plurality of pre-defined styles (e.g., normal, currency, fixed, percent, and the like). The font buttons 226 effect font changes, including toggling bold and italic fonts, as well as increasing and decreasing font (point) size. The insert and delete buttons 227 permit the user to insert or delete blocks, rows, columns, and pages (described in further detail hereinbelow), as desired. The fit button 228 allows the user to quickly tailor a column's width to match its widest cell entry. Action buttons 229 provide automated spreadsheet operations, including

BNSDOCID: <EP___0569133A2_I_>

sorting and summing operations. For example, a Sort button, when invoked, performs a sort on data in a currently active block. A sum button, on the other hand, totals the values in any block of cells by generating an @SUM function, just outside a selected range of blocks.

2. Notebook and pages

Notebook 250, shown in further detail in Fig. 2C, provides an interface for entering and displaying information of interest. The notebook 250 includes a plurality of spreadsheet pages, such as page 251 (Page A). Notebook 250 also includes a "Graphs page" for readily accessing all the graphs created for the notebook. Each page may include conventional windowing features and operations, such as moving, resizing, and deleting. In a preferred embodiment, the notebook 250 includes 256 spreadsheet pages and one Graphs page, all of which are saved as a single disk file on the mass storage 107. In accordance with the present embodiment, workspace 200 may display one or more notebooks, each sized and positioned (e.g., tiled, overlaping, and the like) according to user-specified constraints. When a single notebook is used, notebook 250 will typically occupy a majority of workspace 200.

Each spreadsheet page of a notebook includes a 2-D spread. For example, spreadsheet page 251 includes a grid in row and column format, such as row 252 (row 3) and column 255 (col. F). At each row/column intersection, a box or cell (e.g., cell C4) is provided for entering, processing, and displaying information in a conventional manner. Each cell is addressable, with a selector 253 being provided for indicating a currently active one (i.e., the cell that is currently selected).

As shown in Figs. 2C-E, individual notebook pages are identified by page identifiers 260, preferably located along one edge of the notebook 250. In a preferred embodiment, each page identifier is in the form of a tab member (e.g., members 261a, 262a, 263a) situated along a bottom edge of the notebook. Each tab member may include representative indicia, such as textual or graphic labels, including user-selected titles representing the contents of a corresponding page. In Fig. 2D, the tab members 260 are set to their respective default names. For example, the first three tab members (members 261a, 262a, 263a) are respectively set to A, B, and C. In a preferred embodiment, however, tab members are typically given descriptive names provided by the user. As shown in Fig. 2E, for example, the first three tab members have now been set to "Contents" (tab member 261b), "Summary" (tab member 262b), and "Jan" (tab member 263b). In a similar manner, the remaining tabs are set to subsequent months of the year. In this manner, the user associates the page identifiers with familiar tabs from an ordinary paper notebook. Thus, the user already knows how to select a page or spread of interest: simply select the tab corresponding to the page (as one would do when selecting a page from a paper notebook).

In addition to aiding in the selection of an appropriate page of information, the user-customizable page identifiers aid in the entry of spreadsheet formulas. For example, when entering a formula referring to cells on another page, the user may simply use the descriptive page name in the formula itself (as described hereinbelow), thus making it easier for the user to understand the relationship of the cell(s) or information being referenced.

3. Navigation in a notebook

Referring now to Figs. 3A-C, movement (i.e., location of desired information cells) within a spreadsheet notebook of the present invention is illustrated. To move to different pages in the notebook, the user simply selects the corresponding tab from tabs 260. To move to Page B, for example, the user selects (e.g., with keyboard 104 or pointing device 105) tab 262a; similarly, Page C is reached by selecting tab 263a. Continuing the example, the user may return to Page A by selecting tab 261a. Thus instead of finding information by scrolling different parts of a large spreadsheet, or by invoking multiple windows of a conventional three-dimensional spreadsheet, the present embodiment allows the user to simply and conveniently "flip through" several pages of the notebook to rapidly locate information of interest.

Notebook 250 also includes supplemental tools for navigation between pages, including a tab scroller 271 and a fast-forward button 272. The tab scroller (shown in two different states: 271a and 271b) permits access to identifiers for pages which are not currently visible on the screen device 106. If a desired identifier or tab is not currently in view, th user simply activates the tab scroller 271 to reveal additional tabs. The fast-forward button 272, on the other hand, provides immediate access to the last pages of the notebook, including the Graphs page. As shown in Fig. 3A and B, after invoking the fast-forward button 272a, the page identifiers for the last pages (e.g., tabs 261c, 262c, 263c, 265) are accessible. To switch back to the previously active spreadsheet page, the user may select or click the fast-forward button 272c again. For navigation within a spreadsheet page, horizontal and vertical scrollbars 278, 259 are provided; general features and operations of basic scroller

BNSDOCID: <EP___0569133A2_I_>

or sliders are described in Windows user and programming literature, including Microsoft's *Windows Software Development Kit.*

1. Selections and aggregate operations within a notebook

The selection of desired information cells in the notebook of the present embodiment is now described. For selecting a plurality of information cells, both 2-D blocks (e.g., block 254 of Fig. 2C) and 3-D blocks of cells may be defined. A 2-D block is a rectangular group of one or more cells and is identified by block coordinates, such as the cell addresses of its upper-left and bottom-right corners. Similarly, a 3-D block represents a solid block (e.g., cube) of cells.

A 2-D block is specified by selecting, with mouse 105 or keyboard 104, opposing corners. In Fig. 2C, for example, the block 254 is defined by corner cells **C5** and **F14**. Additional selection examples are illustrated in Figs. 4A-E. For example, column **B** (col. 411) is selected by clicking column heading 410; similarly, row **3** (row 421) is chosen by clicking row heading 420. Selection may be additive (i.e., additional selections are appended to the current ones), as shown by selection of a row 420 and a column 410 in Fig. 4C. To facilitate the selection of all cell members (e.g., block 431), a select-all button 430 is also provided. In addition to these "contiguous" blocks, non-contiguous block selection (e.g., selection of blocks 441, 442) is provided by use of a status key (e.g., CTRL-, ALT-, or SHIFT-) plus a mouse event (e.g., click and drag operations).

Selection of 3-D cell blocks, i.e., cell ranges spanning more than one page, occurs in a similar fashion. To extend the block 254 (of Fig. 2C) into a 3-D block, the user specifies an additional or third dimension by selecting an appropriate page identifier. If the user selects the **D** tab while the block 254 is selected (e.g., with a SHIFT or other status key), the 2-D block is extended into a 3-D block spanning from Pages **A** to **D**. Additional 3-D operations may be performed by utilizing a method of the present embodiment for grouping pages, which will now be described.

Pages may be selected or grouped together, thereby providing a means for changing multiple pages simultaneously. In much the same manner as cells from a spread are grouped into 2-D blocks, a range of pages are grouped by specifying beginning and ending members. As shown in Fig. 4F, a range from Page **A** to Page **K** may be achieved by selecting tabs **A** (261) and **K** (267) from identifiers 260, for example, while depressing a key (e.g., status key). A grouping indicator 268 is displayed for indicating members of a group; groupings may also be annotated with user-specified labels. Once grouped, a page of the group may have its operations (e.g., selection, data entry, and the like) percolate to the other members of the group, as desired. A non-contiguous selection of pages may also be selected (even across different pages); for example, a selection of Pages **A** and **D**, but not **B** and **C**, may be achieved by selecting tabs **A** and **D** while depressing a second key (e.g., CTRL-key). Furthermore, groups may overlap (i.e., a page can be in more than one group), as desired. By selectively activating a group mode (e.g., by toggling group button 273), groupings may be temporarily turned off, in which case events are not percolated to other members of the group.

With group mode active, an activity in a page which is a member of a group can also affect similarly situated cells of the other pages of the group. For example, information entered in a cell on one page of the group can also propagate to the same (relative) cell in other pages of the group; data entry may be "drilled" (propagated) to other group pages by concluding data entry, for example, with a "Ctrl-Enter" keystroke (instead of a "Enter" command). For instance, an entry into cell **C4** of Page **A** may also conveniently and automatically appear in cell **C4** of pages **B, C, D, E, F, G, H, I, J,** and **K**, for an **A- K** grouping. Similarly, block or aggregate operations may propagate across member pages. For example, a block operation (e.g., Block Fill) for cells **A1** to **C4** in Page **A** in an **A-D** grouping would also operate (in this case, fill information) in the same cells (i.e., from **A1** to **C4**) for Page **B** to Page **D**.

Employing the user-specified page identifiers of the present embodiment, a simple nomenclature is available for specifying these solid blocks of information. In a preferred embodiment, a solid block is specified by: <<First Page>> .. <<Last Page>> : <<First Cell>> .. <<Last Cell>>

For example, a solid block may be defined as **A..D:A1..C4**, in which case the block spans from cells **A1** to **C4**, and spans across Pages **A-D**. By permitting alias names (i.e., user-supplied alternative labels), the present embodiment allows the block to be specified as **1989 sales..1992 Sales : A1..C4**; or even **1989 Sales..1992 Sales : First Quarter..Third quarter**. Additionally, the present embodiment readily accommodates notebook information as well, for example, **[TAX]1989 Sales..1992 Sales : First Quarter..Third Quarter**, thus permitting information to be linked across various notebooks. Wildcard characters (e.g., * and ?) may also be employed for cell, page, and notebook identifiers, as desired. Thus, the spreadsheet notebook of the present embodiment provides a 3-D interface which readily accommodates real-world information in a format the user understands (instead of forcing the user to adapt his or her information to fit an arbitrary spreadsheet model).

BNSDOCID: <EP___0569133A2_I_>

5. Advanced Editing

Whether 2-D or 3-D in nature, blocks of cells may be easily copied and cut (i.e., moved) using drag-and-drop editing techniques of the present embodiment. As shown in Fig. 4G for a 2-D block, for example, a method for copying a block of cells includes (1) selecting a source block by dragging a range of cells (e.g., mouse button-down events coupled with mouse movement across the range; close selection with a button-up event), (2) dragging the block (e.g., click within block, followed by repeated mouse button-down events), and (3) dropping the block (e.g., mouse button-up event at desired target location). In a similar fashion, 3-D blocks may be dragged and dropped.

In typical cut and copy operations, relative and absolute cell addressing is employed, as is known in the art (see e.g., *Using 1-2-3*). According to the present embodiment, however, a "model copy" technique for copying blocks is also provided. Model copying, illustrated in Figs. 4H-J, is useful when the user is copying a block that contains absolute references to cells within the copied block. In Fig. 4H, a small spread model 496 is shown which contains a formula to figure the monthly payment for a 30-year loan at different interest rates; a reference to the loan amount was entered as absolute so that when the formula is copied, it continues to refer to cell B1. The user may want to calculate (at the same time) monthly payments for different loan amounts and, thus, might copy the model, with the loan amount changed (shown in Fig. 4I as spread 497). However, in this approach the absolute reference still refers to row 1; to correct this, the user would typically manually edit each formula to refer to **B6** instead of **B1**.

With model copying of the present embodiment enabled (e.g., by default or through a user dialog), however, the problem is solved. In particular, absolute references adjust to the new location of the referenced cell, as shown by spread 498 in Fig. 4J; however, absolute references remain absolute to make future copies absolute. For instance, should the user make more copies of the formula, the reference to cell **B6** is still absolute. In this manner, model copying of the present embodiment saves the user time-consuming and error-prone editing of formulas.

As shown in Fig. 4K, notebook pages may be copied or moved using the drag-and-drop editing techniques of the present embodiment (a dialog box is also provided as an alternative). As shown, for example, the "Salads" page is moved by selecting its tab identifier 264a from the identifiers 260; in a preferred method, the identifier is selected by dragging it downward (mouse button-down plus downward movement) with the mouse cursor. Next, the identifier is dragged to a desired new position (mouse button-down plus left and right movement). Once positioned at a desired location (as indicated by in-transit tab identifier 264b), the page is then "dropped" (mouse button-up) into position. Similarly, a "copy" operation may be effected by coupling the above operation with a status key (e.g., CTRL-); in a preferred method of copying, only the page information (not its tab identifier) is copied to a target location.

Additional editing operations may be easily accomplish using the page identifiers of the present embodiment. For example, an additional page may be inserted by selecting a desired tab and entering "INS" (keyboard or mouse). Similarly, a notebook page may be deleted by coupling the selection of a corresponding page tab to a delete command.

6. Advantages

In contrast to prior art spreadsheet implementations, use of the spreadsheet notebook of the present embodiment is easily ascertained by the user. The notebook interface for example may provide a convenient means for organizing many spreadsheets together into one file. This permits the user to load (into memory 102) all related information with a single command, without having to remember a variety of different file names. Moreover, the notebook interface 250 encourages a user to spread data out among multiple pages, thus better organizing one's data. In Fig. 4L, for example, spreadsheet page 300 illustrates the conventional method for grouping unrelated information onto a single spreadsheet. As shown in column B, the incorporation of unrelated information into a single spread leads to unnecessary whitespace padding of information. In Fig. 4M, in contrast, related information is grouped on separate pages 351, 352 (e.g., with their own columns, Col. B_1 and Col. B_2) of notebook 350. Not only does this eliminate grouping of disparate information, with its disadvantages (e.g., padding data entries), but it also encourages better organization of information.

Inspecting and Setting the Properties of Objects

A. Disadvantages of Prior Techniques

Standard windowing interfaces depend heavily on a clunky system of pull-down menus. While pull-down

BNSDOCID: <EP__0569133A2_I_>

menus are an improvement over command-line (e.g., MS-DOS) interfaces, they are non-metaphoric or non-analogous to ordinary objects, i.e., ones familiar to the user. Thus, prior art windowing GUIs are no more intuitive or useful to the user than other menuing systems.

Perhaps the biggest weakness of pull-down menus is the requirement that the user must know beforehand which menu contains the item or function of interest. If the user does not know which pull-down menu, submenu, or dialog box contains the item he or she is seeking, the user will spend an inordinate amount of time checking the contents of each in an effort to hunt down the needed item. And often the search is in vain. Moreover, since functions for a given object (e.g., text object) are often scattered over several disparate menus, the user is discouraged from interacting and experimenting with the object.

One approach to overcoming this problem has been the implementation of "smart icons." This interface technique employs screen buttons painted with icons which are supposed to tell the user what the buttons do. This approach, however, often makes the interface problem even worse because the meaning of the icons are not easily grasped. Thus, the approach is often no more helpful than hiding the desired function deep inside a menuing system. Worse, some button images are so small or so numerous that it is hard to see the icons well enough to discern what they mean.

B. Property Inspectors

1. Introduction

Overcoming this problem, the present embodiment provides "property inspectors" for inspecting and setting the properties of objects. Property inspectors of the present embodiment examine an object of interest to the user and tell the user what can be done to it. In this manner, the present embodiment requires the system and its interface components (e.g., menus or dialog boxes), not the user, to know what functions are being sought or may be of immediate interest to the user. Moreover, property inspectors of the present embodiment require the program to present functions to the user wren and where he or she requests them, rather than making the user hunt for them when they are needed. Thus, the user need only know which data item or object he or she wants to inspect or manipulate.

In the spreadsheet notebook, for example, there are many kinds of objects which one can use. Examples include cells, blocks of cells, pages, note books, graphs and their elements (including bars or axes), dialog boxes, and even the application program itself. Each of these objecto has properties, which are characteristics of that particular object. For example, blocks of cells have a font property that can be set to bold, so that the text of entries in the block appear in boldface type. A property of a workbook page, on the other hand, is the name that appears on its tab. Each notebook has its own palette property for controlling available colors. Application properties, for instance, include system defaults, such as a default storage directory or file extension. Thus in any system, the user has a variety of objects available, each of which has its own set of properties.

Property inspection of the present embodiment provides the user with immediate access to an object's properties. If the object of interest is a spreadsheet cell, for example, the property inspector of the present invention produces a menu that includes font, color, border, size, and other display properties, along with formula properties which may be modified. If, on the other hand, the object is a graph, the property inspector will display a menu to change its color, shape, borders, and other features of a spreadsheet graph. Moreover, inspection menus are state or context-sensitive, i.e., constructed on-the-fly to reflect the current state of the object. If an object's properties change so that what a user can do to it also changes, the property inspector of the present embodiment will create a new and appropriate menu reflecting those changes.

A variety of user events, including keyboard and mouse events, may be employed to invoke property inspection and setting. In a preferred method of the present embodiment invention, however, an object is inspected by selecting the object with a screen cursor, i.e., clicking (generating a mouse event) on the object. Since, according to the present embodiment property inspection may be available across different modes of operation (i.e., generally available at all times), the generated mouse event or signal is preferably one which does not launch or otherwise invoke routine object actions. Instead, the mouse signal should be one which the user will easily associate with inspection of an object. In a two or three mouse button embodiment therefore, the generated mouse signal is most preferably from the lesser-used or right mouse button (e.g., Windows' WM_RBUTTONDOWN). In this manner, the user associates object actions with left button signals and object inspection with right button signals.

2. Exemplary Embodiments

Referring now to Figs. 5A-E, the inspecting and setting of object properties is illustrated. In Fig. 5A, a note-

BNSDOCID: <EP__0569133A2_I_>

book application window 500 is shown. Window 500 includes a variety of objects which may be inspected by the property inspector of the present embodiment. The top-most object is the application itself: application object 510. By selecting the application object (e.g., by right mouse clicking on the application title bar), various properties for the application may be inspected and set. As shown in Fig. 5B, inspection of the application object 510 invokes application inspector 515, which lists the various properties of the application object; the user may now inspect and set the properties of the object, as desired.

Referring back to Fig. 5A, the next level which may be inspected is the notebook object 520. User inspection of this object, e.g., by right mouse clicking on the notebook title bar, invokes the active notebook property inspector 525 of Fig. 5C. In a manner similar to the application property inspector, the notebook property inspector 525 includes all the properties and corresponding options which may be set for the notebook 520.

Notebook object 520 includes a plurality of page objects, which may themselves be inspected. To inspect the page object 550, for example, the user brings the page tab into view (if it is not already displayed), then right clicks the tab page. This action invokes the active page property inspector 535, as shown in Fig. 5D. With access to page properties, the user can control the page name, overall page protection, line color, colors in cells that meet certain conditions, default label alignment, whether zeroes are displayed, default column width, row and column borders, gridline display, and the like. To assign a more descriptive name to a page, for example, its tab is inspected and the option "NAME" is selected from the page property inspector (or, as shown, it is selected by default). After the user enters a new name, the new name appears on the page tab. Other page properties, most of which affect the appearance of data or screen elements, may be set in a similar manner.

Each page object may also contain other objects, including cell, blocks of cells, graph objects, and the like, each of which may be inspected. By right clicking the active block 530, the active block property inspector 545 is opened as shown in Fig. 5E. With block properties, the user can change alignment of entries, numeric format of values, block protection, line drawing, shading, font, text color, types of data allowed in a cell, row height, column width, and whether columns or rows are revealed or hidden; inspection of a single cell occurs in a similar manner (i.e., active block is one cell). In this manner, the user may select block properties which affect the appearance of cell entries or of rows or columns; additionally, properties which affect the way data can be entered into blocks, such as protection and data entry input, may be made.

Upon invocation of property inspection, a dialog box or property inspector is displayed, preferably at or near the object of interest. A different property inspector appears depending on the type of object which is inspected. At this point, the user may then select the property that he or she wishes to change from a list of object properties. The contents of the right side of the inspector (i.e., a "pane" within the inspector) change to correspond to the property one chooses, as shown. Then, the user chooses settings for the current property. If desired, the user can then choose and set other properties for the current object. In a preferred embodiment, the property inspector includes an example box, which shows the display result of the property the user has just set, before they are applied to the actual object.

Referring now to Figs. 6A-K, the construction and operation of a property inspector dialog box, in accordance with the present embodiment is described. For purposes of illustration, the following description will present active block property inspector 600; other inspector dialogs may be implemented in a similar fashion. Shown in Fig. 6A, the active block property inspector 600 includes an object property list 601 and property settings pane 602. For an active block, for example, relevant properties include Numeric Format, Font, Shading, Alignment, Line Drawing, Protection, Text Color, Data Entry Input, Row Height, Column Width, and Reveal/Hide. Property settings pane 602 include the valid options or settings which a property may take. For the property of numeric format, for example, valid settings include fixed, scientific, currency, and the like. Property setting pane 602 may further include subproperties. For example, the currency setting 602a shown may also include a decimal place setting 602b. In this manner, property inspector dialog 600 presents a notebook dialog: property list 601 serves as notebook tabs (e.g., dialog tab 601a) and property settings panes 602 serves as different pages of the notebook dialog. Thus, the user may access numerous dialog options with the efficiency of a notebook.

Also shown, property inspector dialog 600 also includes an example window 605 and dialog controls 603. As new properties are selected and set, their net effect is shown. The example element in example window 605, for example, shows the effect of selecting a numeric format of currency with two decimal places. Thus, the user may experiment with changes before they are applied to the data object. After desired property changes are entered, control components 603 are invoked to either accept ("OK button") or reject ("CANCEL button") the new property settings; if desired, help may be requested.

As shown in Figs. 6B-K, other properties of a cell block are accessed by selecting the desired property from the property list 601. Thus the active blocks property inspector 600 changes to inspector 620 for font properties, inspector 625 for shading properties, inspector 630 for alignment properties, inspector 635 for line

BNSDOCID: <EP___0569133A2_I_>

drawing properties, inspector 640 for cell protection properties, inspector 645 for text color properties, inspector 650 for data entry input properties, inspector 655 for row height properties, inspector 660 for column width properties, and inspector 665 for reveal/hide properties. In each instance, a new pane (i.e., one having the attributes of interest) is displayed in the inspector window.

Referring now to Figs. 7A-H, the inspection (and setting) of properties for graphs is illustrated. Graph window 700 includes a plurality of graph objects, each of which may be customized through use of a corresponding property inspector of the present embodiment. To display a corresponding property inspector, the user invokes the inspector (e.g., right-clicks) for the part (object) of the graph he or she wishes to change. A right-click on the graph window object 710, for example, will invoke the graph window inspector 715; at this point, the user may inspect and set various properties of the graph window object. In a similar manner, other objects of the graph window may be inspected. For example, inspection of graph background 720 invokes inspector 725, Y-axis object 730 invokes inspector 735, X-axis 740 invokes inspector 745, area fill object 750 invokes inspector 755, bar series object 760 invokes inspector 765, and bar series object 770 invokes inspector 775.

Internal Operations

A. Introduction

Internal operations of the system 100 will now be described in detail. In general, operation is driven by methods which are invoked by Windows' message dispatcher in response to system or user events. The general mechanism for dispatching messages in an event-based system, such as Windows, is known in the art; see, e.g., Petzold, C., *Programming Windows*, Second Edition, Microsoft Press, 1990. Additional information can be found in Microsoft's Window Software Development Kit, including: 1) *Guide to Programming*, 2) *Reference*, Vols. *1 and 2*, and 3) *Tools*, all available from Microsoft Corp. of Redmond, WA. Of particular interest to the present invention are class hierarchies and methods of the present invention, which are implemented in the C++ programming language; see e.g., Ellis, M. and Stroustrup, B., *The Annotated C++ Reference Manual*, Addison-Wesley, 1990. Additional information about object-oriented programming and C++ in particular can be found in Borland's C++ 3.0: 1) *User's Guide*, 2) *Programmer's Guide*, and 3) *Library* Reference, all available from Borland International of Scotts Valley, CA.

B. Notebooks

Referring now to Fig. 8A, the internal structure and operation of spreadsheet notebooks of the present embodiment will be described. The spreadsheet notebook system of the present embodiment includes a system hierarchy 800, at the top level, having a notebook table 805 with entries or slots for accessing a plurality of notebooks. For example, the first slot addresses or points to notebook 810. In this manner, the system may track multiple notebooks.

Notebook structure 810, in turn, includes or accesses various data members for a particular notebook. For example, a "Name" field, stores the name assigned for the notebook. This name is displayed in the titlebar for the notebook and is used as the name for the corresponding disk file; the notebook name is also used to reference the notebook in formulas (e.g., contained in other notebooks). Notebook 810 also includes other data members, such as block names and fonts. Block names are text strings or labels which the user has defined for particular blocks of interest. Fonts, on the other hand, include font information (e.g., type and size) for the notebook. Other desired properties, specific to a notebook, may be included as desired.

Of particular interest in the notebook structure 810 is the Pagetable field, which includes a pointer to a page table for the notebook. Pagetable 815 includes a plurality of slots or entries for accessing individual page structures of the notebook. Each page structure, in turn, contains or accesses information of interest to an individual page. As shown, for example, page 820 (pointed to by the first slot of the pagetable) includes or accesses: a sheet table (or Graphs page), a pagename, floating objects (e.g., graphs and buttons), page properties, pane, and the like.

The Sheettable field of the page 820 points to a sheet table 825. It, in turn, includes a plurality of the slots for accessing different column strips. As shown, for example, the first entry in the sheet table 825 accesses a column strip of cells 830. In turn, column strip 830 addresses individual information cells, each of which may have one of a variety of information types, including empty, integer, number (real), formula, label, and the like. In a preferred embodiment, column strip 830 may address up to 8,000 cells; those skilled in the art, however, will appreciate that column strip 830 may be set to any desired range (depending on the limits of one's target hardware).

Referring now to Figs. 8B-C, the function of the page table of the present embodiment is illustrated. In

BNSDOCID: <EP___0569133A2_I_>

Fig. 8B, two images are presented: a view image and a data image. The view image illustrates what the user sees on the screen 106; at startup, for example, only a single Page A (page 850) is active. As shown by the corresponding data image supporting this view image, pagetable 855 includes only a single reference, page A. In turn, page A references $sheettable_A$, which supports the information cells required (e.g., they are currently needed for display) for the page. Thus, at startup, only a single page entry exists in the pagetable 855, even though the screen displays multiple page tabs.

Upon selection of Page F (e.g., by clicking tab F 860), the data image changes, however. As shown in Fig. 8C, remaining Pages B-F are initialized into the page table 865. In this example, Page F must now exist as a view object. Thus, Page F references supporting data structures (e.g., $sweettable_F$). The intervening Pages (B-E), on the other hand, are initialized, but only to the extent that they can serve as place holders in the pagetable 865. In this manner, the underlying page table for a notebook instantiates underlying page data structures only as needed (e.g., for viewing or referencing by other pages), but, at the same time, provides on-demand access to pages.

A particular advantage of this design is the ease in which information may be referenced and presented. Even complex data models, such as those spanning multiple dimensions, may be represented in a clear fashion. Moreover, a syntax is provided by the present embodiment for intuitively referencing information.

Referring now to Figs. 8D-F, the referencing of information in a spreadsheet notebook of the present embodiment is now described. Shown in Fig. 8D, a three-dimensional reference (i.e., identifier for a solid block of information cells) includes a notebook, starting page, ending page, starting cell and ending cell. As shown, the notebook (which is the same as the file name) is identified preferably by delimiters, such as brackets ([]). This is followed by page name(s), which may in fact be grouped. As shown, a range of pages has been defined from **'89 Income** to **'92 Income**; these are alias names which may correspond to pages A-D, for example.

Next, one or more cells of interest are identified. For example, a range from January to (..) **December** is shown, which serve as aliases for blocks **A1** to **A12**, for example. The block or cell information is separated from group or page information, for example, by a colon (:) identifier.

The hierarchy of this nomenclature is shown in Fig. 8E. Specifically, a notebook references pages, which may be members of one or more user-defined groups. In turn, a page references cells, which may alternatively be referenced by alias identifiers. As shown in Fig. 8F, page and cell identifiers may be grouped to form even more succinct references. For example, Pages **'89 Income** to **'92 Income** may be renamed as **Four-Year Income**. Similarly, the cell range from **January** to **December** may be renamed **Entire Year**. In this manner, information ranges in the spreadsheet notebook of the present embodiment. are easily named and easily visualized by the user.

Depending on the context of the system, certain identifiers may be assumed and, thus, eliminated from a reference. Unless otherwise specified, for example, the notebook identifier is assumed to be the currently active notebook. Similarly, the page identifier may be assumed to be the currently active page, unless the user specifies otherwise. Thus, a valid reference need only include as much information (identifiers) as is necessary to access the desired information.

Referring now to Fig. 9A, a method for interpreting or parsing an information reference, in accordance with the present embodiment is shown. Upon receiving a reference (e.g., text string) to information in a spreadsheet notebook, the method proceeds as follows. In step 901, a loop is established to tokenize (i.e., parse) the string into individual identifiers, which may then be processed. Each token is examined as follows. In step 902, the token is checked to determine whether it is a notebook identifier. In a preferred method, a notebook identifier is delimited, for example, by bracket symbols; alternatively, a notebook identifier may be realized simply by the position of the identifier relative to other tokens in the string, or by its relationship to disk (notebook) files on mass storage 107. If the notebook is successfully identified, the notebook identifier is returned at step 903, and the method loops back to step 901 for any additional tokens.

If it is not a notebook (no at step 902), however, then in step 904 the token is examined to determine whether it is a group name. Group names may be determined by accessing group names listed for the notebook (e.g., by accessing a group name string pointer from notebook structure 810), and/or by the token's context (e.g., preceding a ":" delimiter). If a group name is identified, it is returned at step 905, and the method loops for any remaining tokens. Otherwise (no at step 904), the token is examined to determine whether it is a page. In a manner similar to that for checking group names, a page may be identified by examining a notebook's page table, with corresponding page name accessed (dereferenced). The page is returned at step 907, with the method looping back to step 901 for remaining tokens.

If a page is not found (no at step 906), however, then at step 908 the token is examined to determine whether it defines a range. A range may include a named block of cells (e.g., "entire year") or, simply, two cell addresses separated by an appropriate identifier (e.g., **A1..B1**). If a range is found, then it is returned in step 909, with the method looping for any remaining tokens. Otherwise (no at step 908), the identifier is examined to

BNSDOCID: <EP__0569133A2_I_>

determine whether it is a cell. A token may be identified as a cell if it is of the appropriate column/row format (e.g., A1). If a cell is found, it is returned at step 911, with the method looping for any remaining tokens.

As shown (conceptually) at step 912, if any ambiguities remain, they may be resolved according to an order of precedence; for example, notebook > groupname > page and the like. At the conclusion of the method, a reference, if it is in proper form, will be successfully identified and may now be processed according to three-dimensional techniques.

Referring now to Figs. 9B-C, other methods of the embodiment are illustrated. In Fig. 9B, for example, a drag-and-drop edit method 920 is shown. Its steps are as follows. In step 901, a block of cells is defined (e.g., by dragging a mouse cursor from one corner of the block to another). After a block has been selected, it is next grabbed with the mouse cursor (i.e., position the mouse cursor within the block and depress a mouse button). In steps 923 and 924, a loop is established to "drag" the block to a new location. In particular, while the mouse button is depressed, any movement of the mouse will cause the block to follow (animated on screen, e.g., by XOR technique). At step 925, the block is "dropped" into place by releasing the mouse button (button up signal). Dropping the block causes information from the source location to be copied into the target location. By coupling the method with a status key (e.g., SHIFT- or CTRL-), the technique supports either "cut" (erase original) or "copy" (leave original) modes of operation. Thus, for example, if a status key is not active as step 926, then in step 927 the original (at the source location) is deleted. Otherwise (yes at step 926), the original remains at the source location as well.

Referring now to Fig. 9C, a model copy method 940 is illustrated in step 941, a block is defined or selected (e.g., dragging a selection). In step 942, model copy is enabled or disabled (as desired); alternatively, model copy may be enabled by default. In step 943 if model copy has been enabled, then in step 945 absolute address references are copied as if they were relative address references, as previously described (with reference to Figs. 4H-J). However, the address labels will remain absolute, so that they will be treated as absolute for future copying operations. Otherwise (no at step 943), absolute addresses are treated conventionally (i.e., referencing absolute addresses) in step 944. As shown in step 946, relative addresses are not affected, i.e., they continue to be treated relatively. In step 947, the copy operation is performed, employing the addresses as just determined, after which the method concludes.

C. Property Inspection

Referring now to Figs. 10A-C, construction and operation of property inspection in accordance with the principles of the present embodiment will be described. As shown in Fig. 10A, user interface (UI) objects are constructed from a UI object class hierarchy 1000. As shown, class hierarchy 1000 includes as its base class an object class 1001. From object class 1001, a responder class 1002 is derived. As the child of class 1001, responder class 1002 inherits all properties of its parent and adds event handling capability. Both object class 1001 and responder class 1002 are abstract classes, i.e., objects are not instantiated from them directly. From responder class 1002, two child classes are derived: view class 1003 and window class 1004. From view class 1003, all screen objects (e.g., text, graphs, scrollbars, and the like) are created. From window class 1004, container objects are instantiated; in particular, window class 1004 provides container objects which hold the various view objects.

Associated with each class (and thus objects derived from each class) is a ClassInfo structure. ClassInfo, which is itself implemented as a separate class, contains information about a corresponding class and, thus, objects of that class. For example, it contains information about object type, name, number of properties, and the like. Of particular relevance to property inspection are two data members: a pointer to the parent and a property record pointer which points to an array of property records for an object of interest.

Referring now to Fig. 10B, the relationship between parent and child (and hence the importance of the pointer to the parent) is illustrated. In the system of the present embodiment an object-oriented mechanism is employed whereby new objects may be created from existing objects (classes). As shown, for example, a child object 1030 may be derived from a parent object 1020. Also shown, each object has its own property record array or list. For example, the parent object 1020 has a property list 1025 describing its properties. Child object 1030, in turn, also has its own property list 1035; child object 1030 still inherits from parent object 1020, however. By means of the parent pointer, the child object 1030 also has direct access to its parent 1020 and, thus, the property list 1025 of the parent. In this manner, when an object is inspected, the system may view back across the inheritance hierarchy to fetch all (ancestor) properties for the objet, as needed.

The property record, on the other hand, describes an individual property for the object. The property record, which is implemented as a separate class, includes a name ID for the property and a pointer to the property object itself (which may be shared). For example, property record objects may be instantiated from the following exemplary C++ class definition:

BNSDOCID: <EP___0569133A2_I_>

```
class _EXPORT_ PropRecord
{
public:
      Property * pProp;       // pointer to SHARED (!) property object
      WORD        nameID;     // property name string ID
                              // (PROPSTR.HPP, PROPSTR.CPP)
      WORD        flags;      // optional information about the property
                              // ( HIDDEN, USE MENU, LINK ONLY, etc.)

      inline Property * getProp ();
      inline LPSTR getName ();
      inline PROPTYPE getType ();
      inline WORD getNamesCnt ();
      inline WORD getNameID ();
      inline WORD getFlags ();
};
```

The property object, in turn, includes a type ID for identifying the property and also includes a pointer to a dialog for the property. An property object may be instantiated from the following exemplary C++ class:

```
class _EXPORT_    Property
{
public:
      static DWORD dwPropErr;
      static char far conversionBuffer [MAXPROPSTR];
      PROPTYPE typeID;
      WORD namesCnt;
      LPSTR pDialog;

      Property ( WORD id );

      virtual BOOL stringToValue ( LPSTR, PROPTYPE pt = 0 );
      virtual BOOL valueToString ( LPSTR, PROPTYPE pt = 0 );

/*
Convert the passed binary value to string using the conversionBuffer
*/
      LPSTR convertToString ( LPSTR pS, PROPTYPE pt );
/*
Convert the string in conversionBuffer to binary value pointed by pS
*/
      BOOL convertToBinary ( LPSTR pS, PROPTYPE pt );
```


BNSDOCID: <EP___0569133A2_I_>

```
}}

Exemplary property types may include:

        prop_Undefined = 0,
        prop_Text,
        prop_Bool,
        prop_Window,
        prop_Color,
        prop_Bitmap,
        prop_List,
        prop_Word,
        prop_Int,
        prop_Key,
        prop_Double,
        ...
```

The pointer to the property dialog (LPSTR pDialog), on the other hand, indicates the appropriate dialog to be displayed to the user for the current property; the dialog displays the appropriate information (e.g., pane) for the property which the user may then use for inspecting and setting attributes of that property.

Referring now to Fig. 10C, a method 1050 for property inspection is illustrated; additional disclosure is provided for the inspection of an exemplary object: a spreadsheet cell. In step 1051, the user indicates that he or she desires property inspection for an object. In a preferred method of the invention, the user will right-mouse click on the object of interest (as set forth hereinabove). To inspect a spreadsheet cell, for example, the user would position the mouse cursor over the cell and click the right mouse button. In this instance, the notebook which contains that cell will receive (i.e., will be the recipient of) a right-mouse button down message. The event will be processed by routing the message to a RightMouseDown method.

In step 1052, the object which the user clicked on is identified by the RightMouseDown method. Specifically, the method invokes a RouteMouseEvent method which returns the current view object (i.e., the object which the user has clicked on). This is accomplished as follows. Every view object contains a StartInspect method which returns an actual object for inspection. The object which appears on the screen to the user (i.e., view object) is not necessarily the object which will be inspected. For example, it would be computationally too expensive to have each cell of a spreadsheet be its own object. Instead, the present invention embraces the notion of surrogate or imaginary objects which may be inspected (using the properties of the screen or view object selected). In most instances, the StartInspect method will return an object for inspection which is the same as the view object. In those instances where it is not feasible to inspect an object which the user perceives on a screen, however, the method returns a surrogate object whose properties assume the attributes of the screen object.

An additional example illustrates this point. When inspecting a block of cells, for example, StartInspect returns a current block object which is a surrogate object, i.e., it does not have a visible expression on the screen. Instead, it is a substitute object which assumes the characteristics necessary to allow inspection of the screen object of interest. In this manner, it functions as a surrogate for the object which the user observes on the screen. If desired, objects may also be blocked from inspection at this step (e.g., by setting flags); in which case, the method terminates. At the conclusion of step 1052, the object of interest, either actual or surrogate, is ready for inspection.

Next, the user interface for inspection is built by a DoUI method. The method proceeds as follows. The first property record for the object is accessed in step 1053. Preferably, DoUI is a virtually defined method, with each object (class) designed to include method steps for accessing the property record according to that object's own particularities. The remaining property records for the object are also located (e.g., by traversing a list of records).

At step 1054, the dialog panes for each property are loaded (e.g., into memory 102, if not already present) for use by a compound dialog box. As previously described, the dialog associated with each property is accessible from the property record (by a pointer to the dialog). At this point, an empty property inspection window is displayed. Into the property inspection window, a corresponding pane for each property is loaded (substep 1054b), thus creating a display hierarchy. Using a getProperty method, each property is initially set to the attribute(s) contained in the retrieved property record; the getProperty method operates essentially the reverse of a setProperty method, which is described in detail hereinbelow. An associated screen button or label is provided for rapidly accessing a desired pane in the inspector window, in much the same fashion as one

BNSDOCID: <EP__0569133A2_I_>

accesses pages in a notebook with tabs. In this manner, a property inspector is built from a dynamic list of properties (as opposed to a static list of properties which is determined beforehand), each of which may be rapidly accessed by the user.

While the property list loaded into an inspector window for each object is dynamically constructed, the panes supporting each property may be pre-built. For example, the pane supporting a color palette, since it is complex in design, will be built in advance.

However, the method may dynamically build individual panes as well. Candidates for dynamic pane building include simple properties, especially those which may have only a Boolean or yes/no value. Referring back to Fig. 7B, for example, inspector 715 includes a snap-to-grid property. Instead of loading a preconstructed pane for this property, the method dynamically constructs an appropriate pane (in this case, a simple check box), as shown in substep 1054a. An automatic list, on the other hand, is typically a simple group box (i.e., one having a plurality of radio buttons), which is preferably constructed dynamically, as shown by the inspector window 650 of Fig. 6H. In either case, the method may build appropriate inspector dialog components on-the-fly by simply examining a particular property and discerning its possible attributes. In a similar manner, "pick" lists of properties may be constructed and displayed, as shown in substep 1054c. A property pick list serves as an index to other property lists or other functions, as desired. By dynamically building inspectors for simpler properties, the method conserves system resources.

Construction of the inspector window is completed by inserting dialog controls (e.g., "OK", "CANCEL", and "HELP") for operating the dialog. In addition, an example window is displayed for indicating various changes to the properties of an object. This is accomplished by sending a message to the inspected object setting forth the current properties in the dialog; the inspected object returns an example or sample object which may then be displayed in the window. After changing a property, the dialog tab or button (e.g., tab 601a of Fig. 6A) corresponding to that property is updated (e.g., different font or screen color) for indicating that a change has been entered.

After constructing the property inspector dialog or window, at step 1055 the method enters a user event loop where the user inspects and sets various properties of the object by accessing the property (through the screen button or tab) and setting a new attribute. At the conclusion of the user event (e.g., by selecting "OK" or "CANCEL"), the user event is terminated.

At step 1056, the property list for the object being inspected is updated for any changes which occurred to the properties during the user event (of step 1055). This is accomplished by calling a setProperty method, which conceptually loops through each pane which has changed and updates the property list, accordingly. By way of illustration, the setProperty method may be defined as: virtual BOOL setProperty(LPSTR lpPropStr, LPSTR lpValueStr, PROPTYPE pt = 0);
The method receives the name of a property, either passed as a text string (lpPropStr) or as a handle (pt), and a value for that property, typically passed as a text string (lpValueStr).

Referring now to Fig. 10D, the general steps of a setProperty method 1070 are illustrated. In step 1071, the property text string is translated; alternatively, the property is referenced directly by passing a handle (i.e., integer or word value). At steps 1072-1073, the property is updated for the property value passed (e.g., by a CASE statement construct). If the property is not acted upon, it may be passed up the object's inheritance chain for action by an ancestor object, at step 1074. In this fashion, the values collected from the various property panes are used by the method to set the properties within the object.

While the foregoing outlines the general steps or framework for setProperty, the method is in fact defined as a virtual method. Thus, each class of objects may assume responsibility for setting its own properties. For an Annotate object (i.e., screen objects found in Graph windows), for example, an exemplary setProperty method may be constructed as follows:

BNSDOCID: <EP___0569133A2_I_>

```
BOOL Annotate::setProperty ( LPSTR lpPropStr, LPSTR lpValueStr, PROPTYPE
pt )
{
      WORD w;

      if ((w = info.propIndex (lpPropStr, pt)) < annoEndProps) {
            if (info.ppProps [w].getType() > prop_Compound)
                  switch (w) {
                        case annoFrame:
                              getFrame (GlobalFrameProp.x,
GlobalFrameProp.y,
                                    GlobalFrameProp.w,
GlobalFrameProp.h);
                              break;
                  }
            if ((info.ppProps [w].getProp())->stringToValue (lpValueStr,
pt)) {
                  BOOL f = TRUE;
                  switch (w) {
                        case annoFrame:
```


BNSDOCID: <EP___0569133A2_I_>

```
                                    s tFrame (Gl balFrameProp.x,
GlobalFrameProp.y,
                                          GlobalFramePr p.w,
GlobalFrameProp.h);
                                    break;
                              case annoHidden:
                                    if (GlobalYesNoProp.index)
                                          hide (TRUE);
                                    else
                                          show (TRUE);
                                    break;
                              case annoShow:
                                    if (GlobalYesNoProp.index)
                                          show (TRUE);
                                    else
                                          hide (TRUE);
                                    break;
                              case annoId:
                                    if (pCW && pCW->infoPtr()->objType ==
ot_Dialog)
                                          f = ((DialView
*)pCW->getContentView())->setIdx (this, GlobalWordProp.w);
                                    else
                                          idx = GlobalWordProp.w;
                                    break;
                              case annoAttach:
                                    flags.nochild = !GlobalYesNoProp.index;
                                    break;
                              case annoPosition:
                                    *(WORD *)&flags = (*(WORD *)&flags
                                          & -(vfPOSITION | vfLEFTREL |
vfTOPREL | vfRIGHTREL | vfBOTTOMREL
                                          | vfTOPTYPE | vfRIGHTTYPE)) |
GlobalPositionProp.flags;
                                    flags.bottomType = flags.topType;
                                    flags.leftType = flags.rightType;
                                    break;
                        }
                        if (f)
                              callConnection (lpPropStr, lpValueStr, pt);
                        return f;
                  }
                  else
                        return FALSE;
            }
            else
                  return Tracker::setProperty (lpPropStr, lpValueStr, pt);
      }
```

As shown, the override method processes properties specific for Annotate (e.g., change frame, ID, position, and the like). In the event that a property is not identified, the property is passed up the inheritance chain for appropriate processing (by a parent's setProperty method). In this manner, an individual object (e.g., an Annotate object) in the system is designed to know how to get (getProperty) and set (setProperty) its own properties.

After the update of step 1056, the method 1050 concludes. At this point, the system is ready for another inspection. Alternatively, the user may undertake other activities, as desired.

While the invention is described in some detail with specific reference to a single preferred embodiment and certain alternatives, there is no intent to limit the invention to that particular embodiment or those specific alternatives. For example, the property inspection of the present invention has been illustrated with mouse devices. Those skilled in the art, however, will appreciate that other input devices, including trackballs, joysticks, light pens, and the like, may be employed. Moreover, the present invention may be advantageously implemented in a variety of UI platforms other than Windows, including Macintosh, X-Windows, Motif, NextStep, OS/2, and the like.

BNSDOCID: <EP___0569133A2_I_>

Claims

1. A computer system providing a user interface between an operator and the system, the system comprising:
   a computer (100) having a processor (101) and a memory (102);
   means (152) for defining addressable cells stored in the memory for entering and processing data and formulas; and
   means (106, 153) for displaying the cells as a two-dimensional spread,
   characterised by:
   means (153) for presenting the cells as a plurality of two-dimensional spreads; and
   the means (106, 153) for displaying being arranged to display the two-dimensional spreads in a notebook format, each spread being represented as a page in a notebook, and each spread including a tab identifier for identifying and accessing the spread.

2. A system according to claim 1, and arranged to display the tab identifiers along one side of the displayed notebook.

3. A system according to claim 1 or 2, and arranged to display a desired spread upon selection of its tab identifier by a user.

4. A system according to claim 1, 2 or 3 wherein each tab identifier includes a label uniquely identifying its spread.

5. A system according to claim 4, wherein cells in a first spread may be referenced in a formula of a second spread by including the label for the first spread.

6. A system according to claim 4 or 5, wherein information in cell(s) of a source spread is referenced in a target spread by a general format of:
<source spread label>:<information cell address(es)>.

7. A system according to claim 4, 5 or 6, wherein the label includes user-supplied alphanumeric information.

8. A system according to any one of the preceding claims and comprising a user-operable pointing device for accessing a spread by pointing at its tab identifiers.

9. In a computer system comprising a central processor coupled to memory, a display, and a user input device, a method of interfacing with data in spreadsheet form, the method comprising the step of:
   (a) displaying a first page array of rows and columns of cells on said display, said cells being arranged to display data input to said memory means;
   characterised by:
   (b) simultaneously with step (a), displaying page indicators for second and third page arrays of cells;
   (c) inputting a signal from said user input device, said signal indicating selection of said page indicator for said second page array of cells; and
   (d) displaying said second page array of cells while simultaneously displaying page indicators for said first and third page arrays of cells.

10. A method according to claim 9, wherein said page indicators are displayed along an outer portion of said display and are formed in the shape of notebook tabs.

11. A method according to claim 9 or 10, wherein said step (b) also displays a page indictor of said first page array of cells, said page indicator of said first page array of cells displayed in a different format from said page indictors for said second and third page arrays of cells.

12. A method according to claim 11, wherein said different format is a different color.

13. A method according to any one of claims 9 to 12, wherein said page indicators are initially marked with default labels and further comprising the step of receiving user input to change said labels.

14. A method according to any one of claims 9 to 13, and comprising the steps of receiving input for page indicator scrolling, and scrolling said page indicators to previously undisplayed page indicators.

BNSDOCID: <EP___0569133A2_I_>

15. A method according to any one of claims 9 to 14, wherein said user input device is a mouse, and said signal is input when said user moves a cursor on said display with said mouse to said indicator for said second page array of cells and clicks said mouse.

16. A method according to any one of claims 9 to 15 and comprising the step of specifying a three dimensional block of said cells for processing, said step of specifying a three dimensional block comprising the step of:

identifying a two dimensional block of cells in said first page array; and

identifying at least one of said second or third page arrays of cells by indicating at least one of said second or third page indicators, cells in said at least one of said second or third pages corresponding to said two dimensional block being identified as being in said three dimensional block.

17. A method as according to claim 16, and comprising the step of copying said three dimensional block to another region of said first page array.

18. A method according to any one of claims 9 to 17, and comprising percolating operations between a selected group of pages by the step of identifying said first and second pages as being in a first group, and performing common operations in cells in said first and second pages, but not in said third page.

19. A method according to claim 18, and comprising percolating operations between a second selected group of pages by the step of identifying said second and third pages as being in a second group, and performing common operations in cells in said second and third pages, but not in said first page.

20. A method according to any one of claims 9 to 19, and comprising the steps of:

labelling a block of cells in said first page array of cells, at least one of said cells in said block being identified as absolute in a model;

copying said block of cells to a second location with said at leat one cell labelled as absolute in a model, cells in said block having cell formulas adjusted to a new address of said at least one block; and

copying said block of cells in a second location without having cell formulas adjusted to a new address of said at least one block.

21. A method according to any one of claims 9 to 20 and comprising the step of inserting an additional page array of cells by specifying a page indicator, and inputting a signal for addition of said additional page array of cells.

22. A computer system providing a user interface between the operator and the system in respect of stored data files, the system comprising a computer (100) having a processor (101) and a memory (102), means (151, 152, 153) for accessing a file, storing portions of the file in the memory and for entering and processing data therein, and means (106, 153) for displaying the data in the accessed file,

characterised in that there are means (153) for presenting each file as a plurality of two-dimensional data areas, and the display means (106, 153) are arranged to display the data areas of the accessed file in the visually perceived form of respective pages of a notebook, each page bearing a tab identifier for identifying the page and by means of which the page can be accessed.

BNSDOCID: <EP___0569133A2_I_>

BEST AVAILABLE COPY

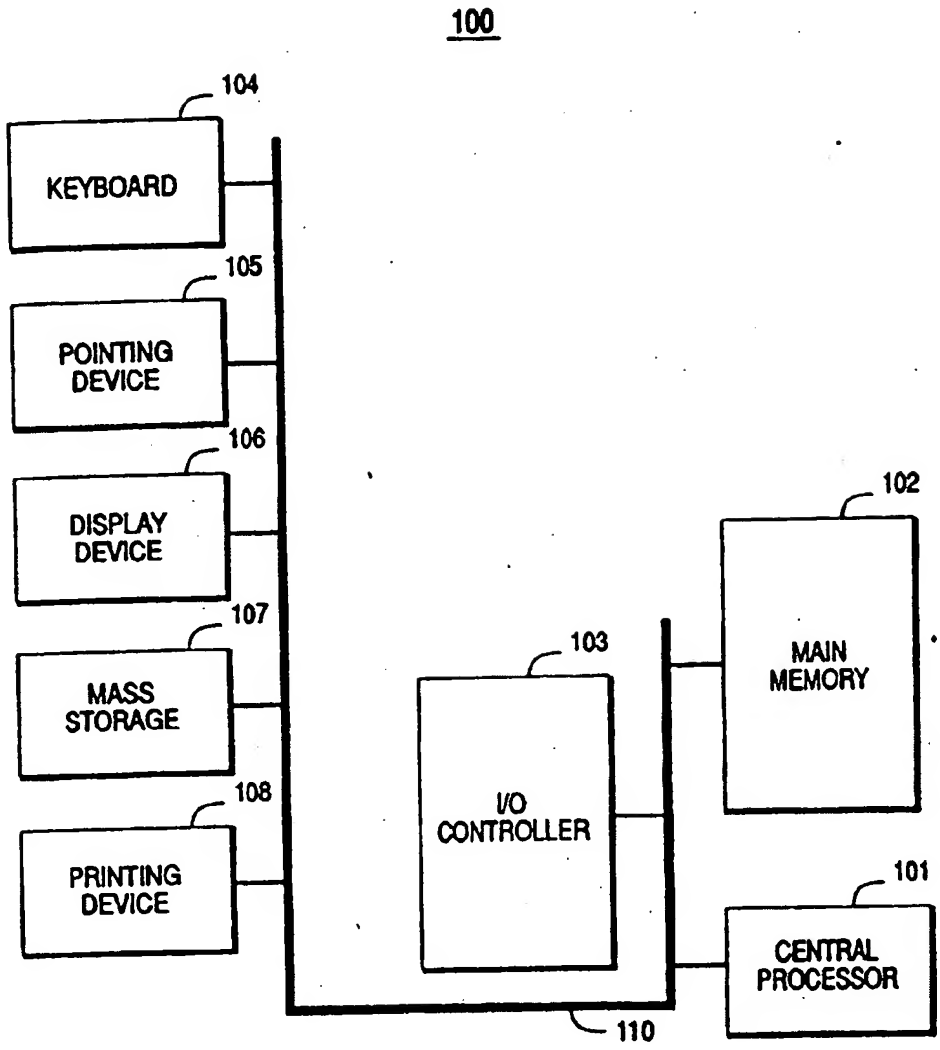


FIG. 1A

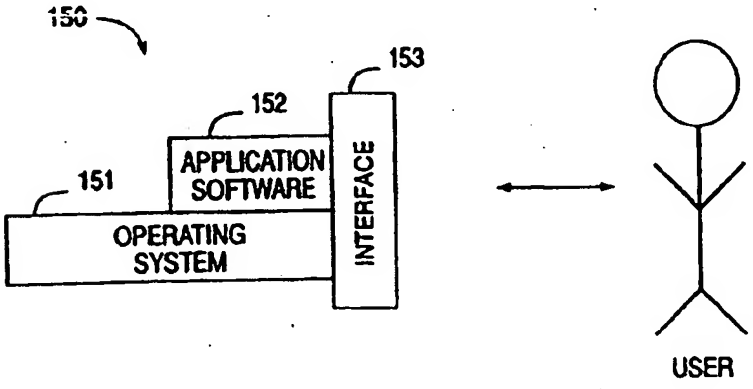


FIG. 1B

BNSDOCID: <EP___0569133A2_I_>

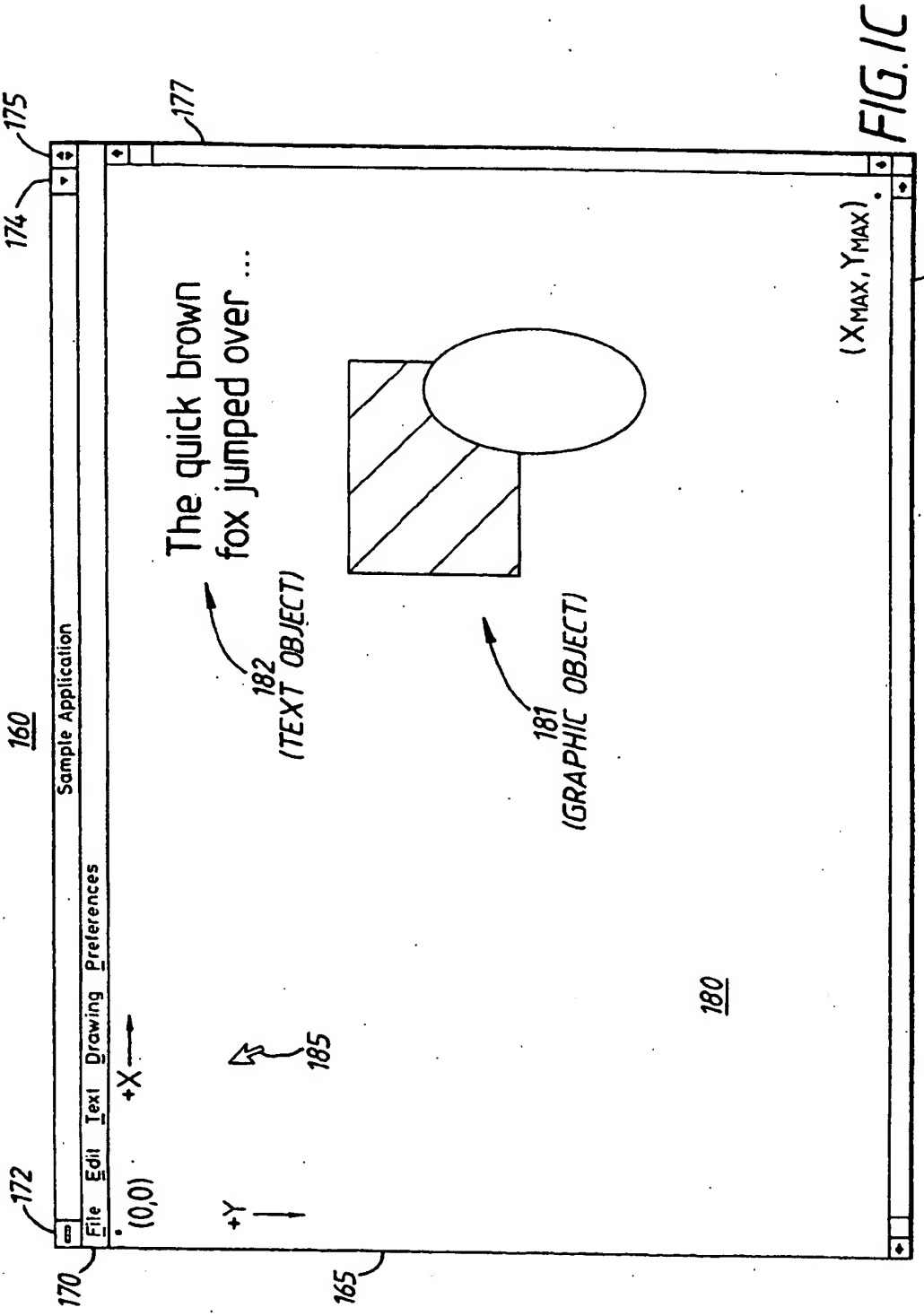


FIG. 1C

BNSDOCID: <EP___0569133A2_I_>

BEST AVAILABLE COPY

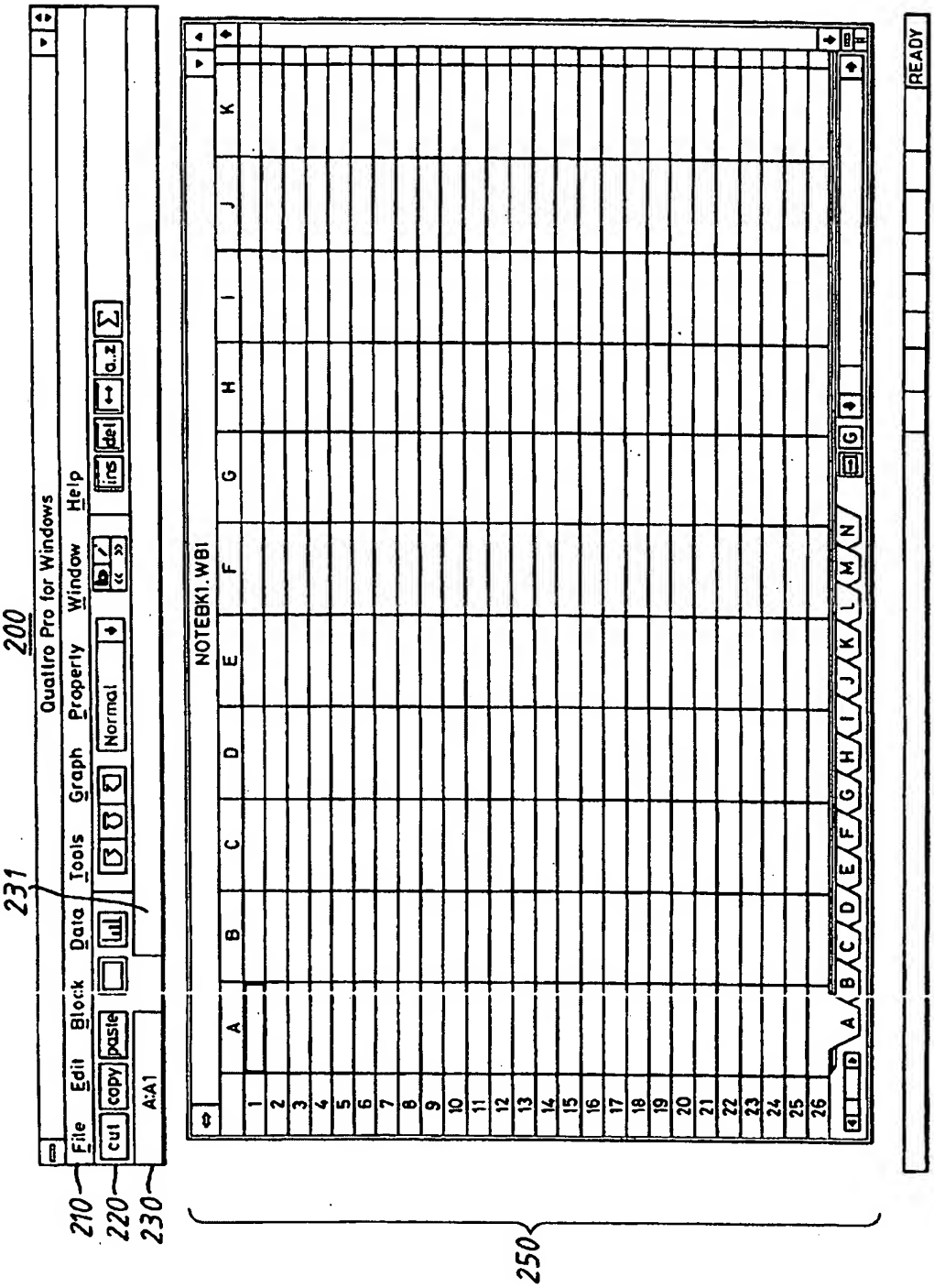


FIG. 2A

BNSDOCID: <EP___0569133A2_I_>

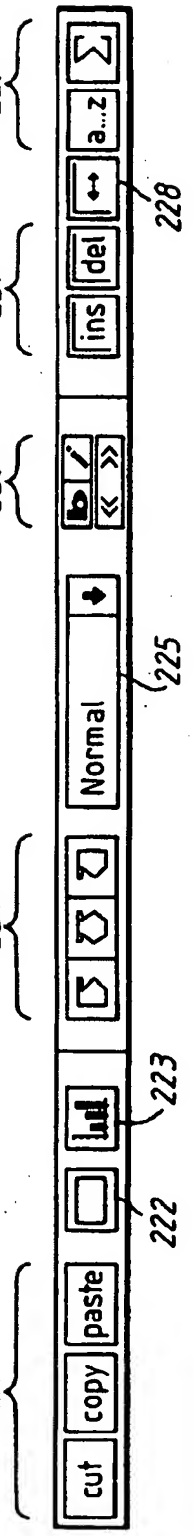


FIG. 2B

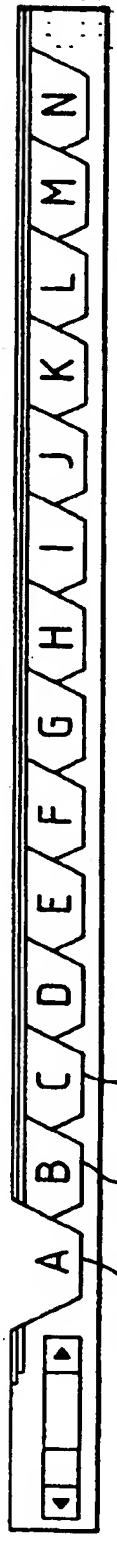


FIG. 2D

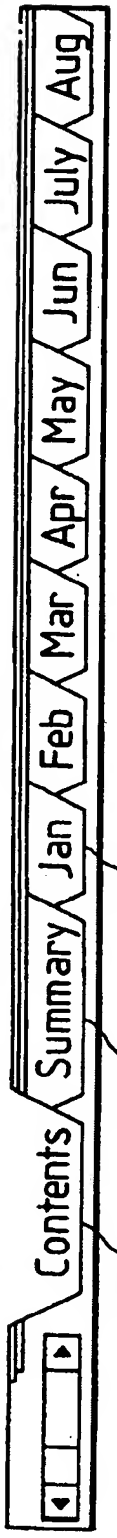


FIG. 2E

BNSDOCID: <EP___0569133A2_I_>

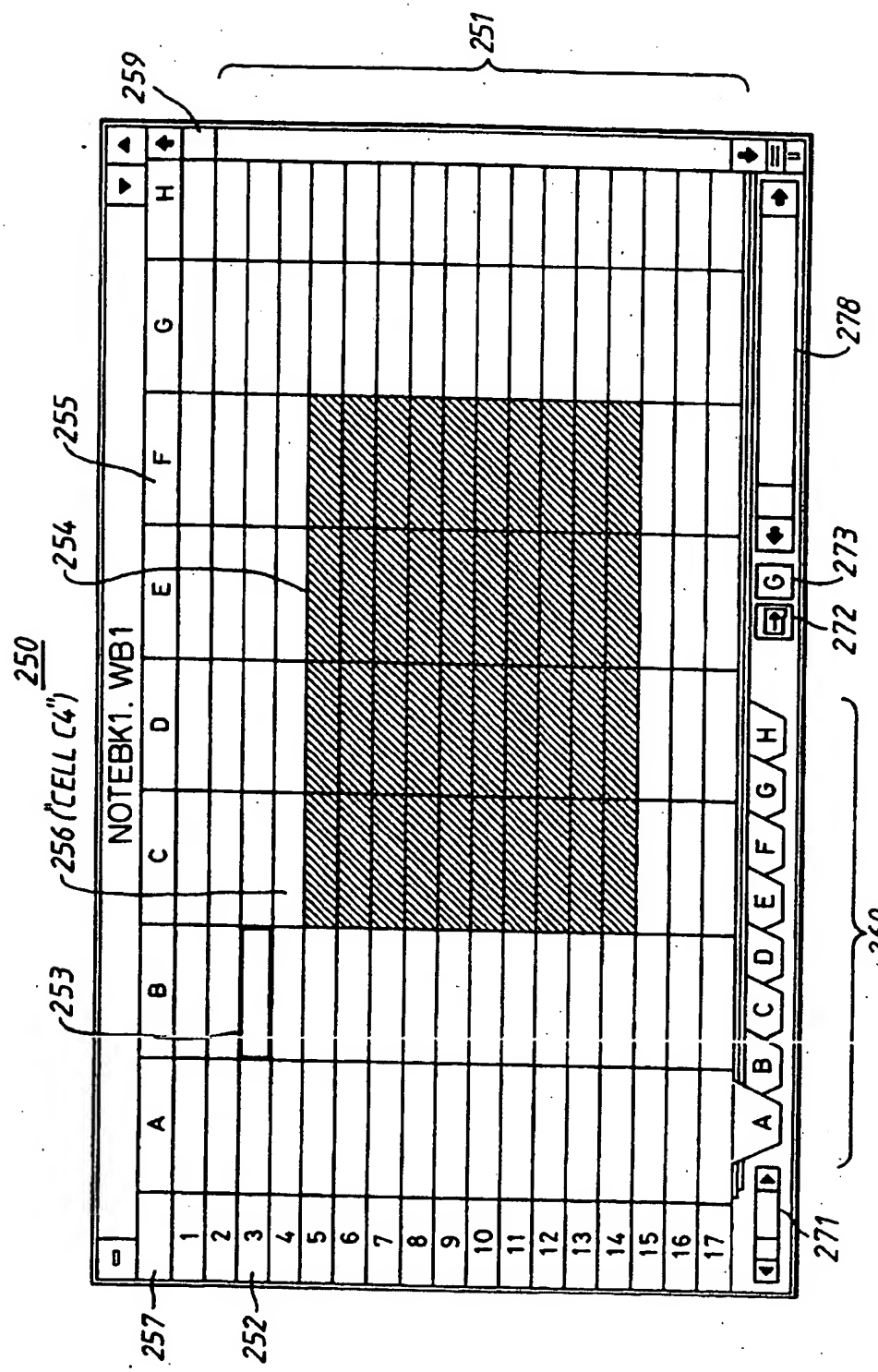


FIG. 2C

BNSDOCID: <EP___0569133A2_I_>

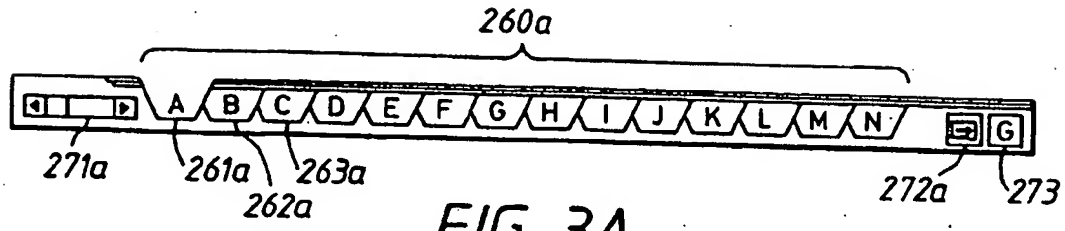


FIG. 3A

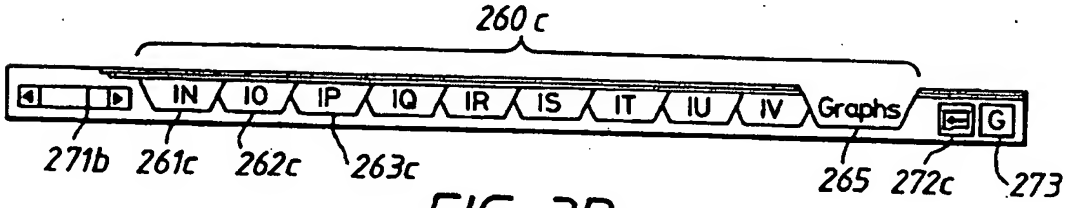


FIG. 3B

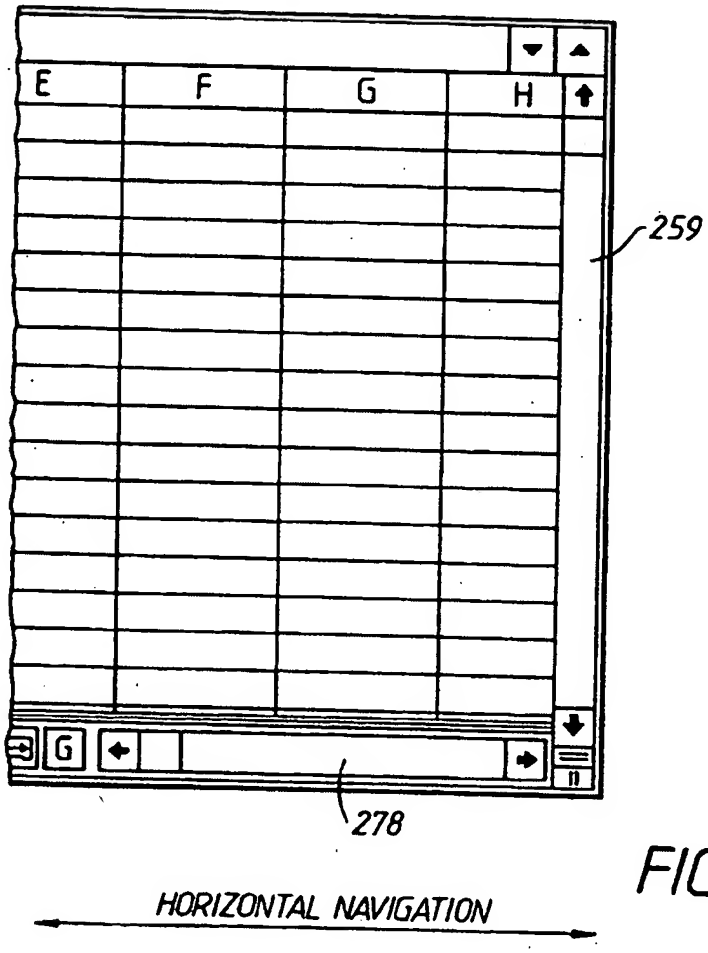


FIG. 3C

BNSDOCID: <EP___0569133A2_I_>

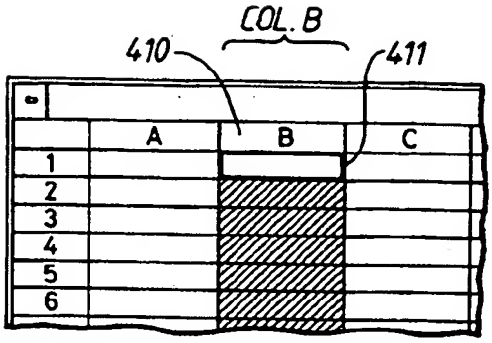


FIG. 4A

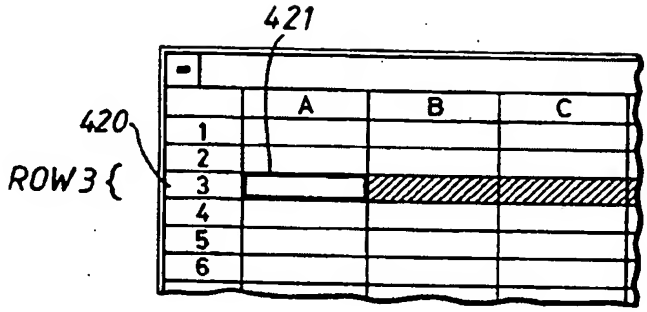


FIG. 4B

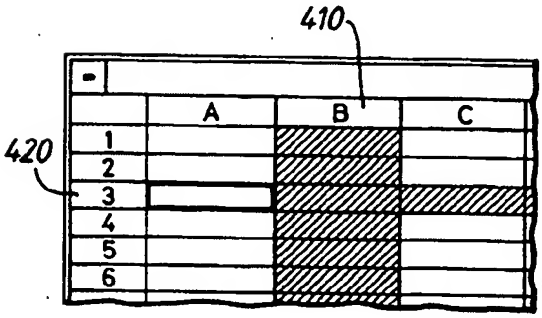


FIG. 4C

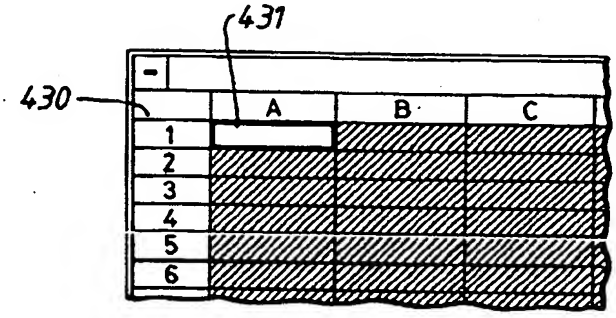


FIG. 4D

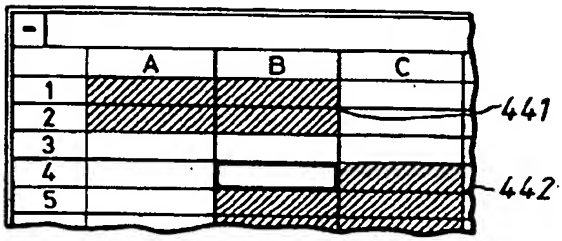


FIG. 4E

BNSDOCID: <EP___0569133A2_I_>

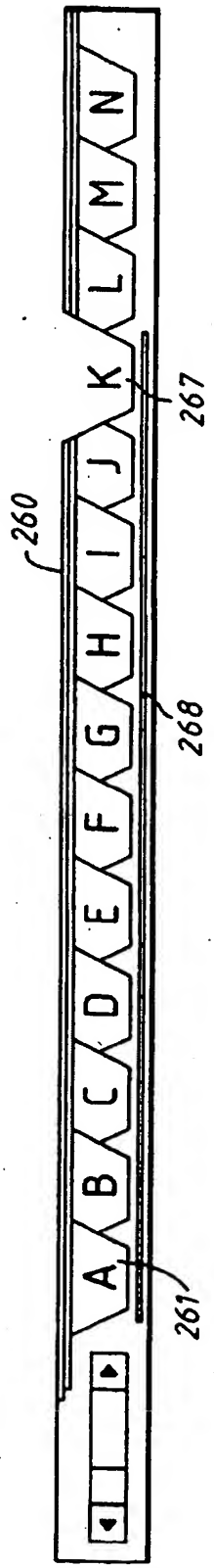


FIG. 4F

BNSDOCID: <EP___0569133A2_I_>

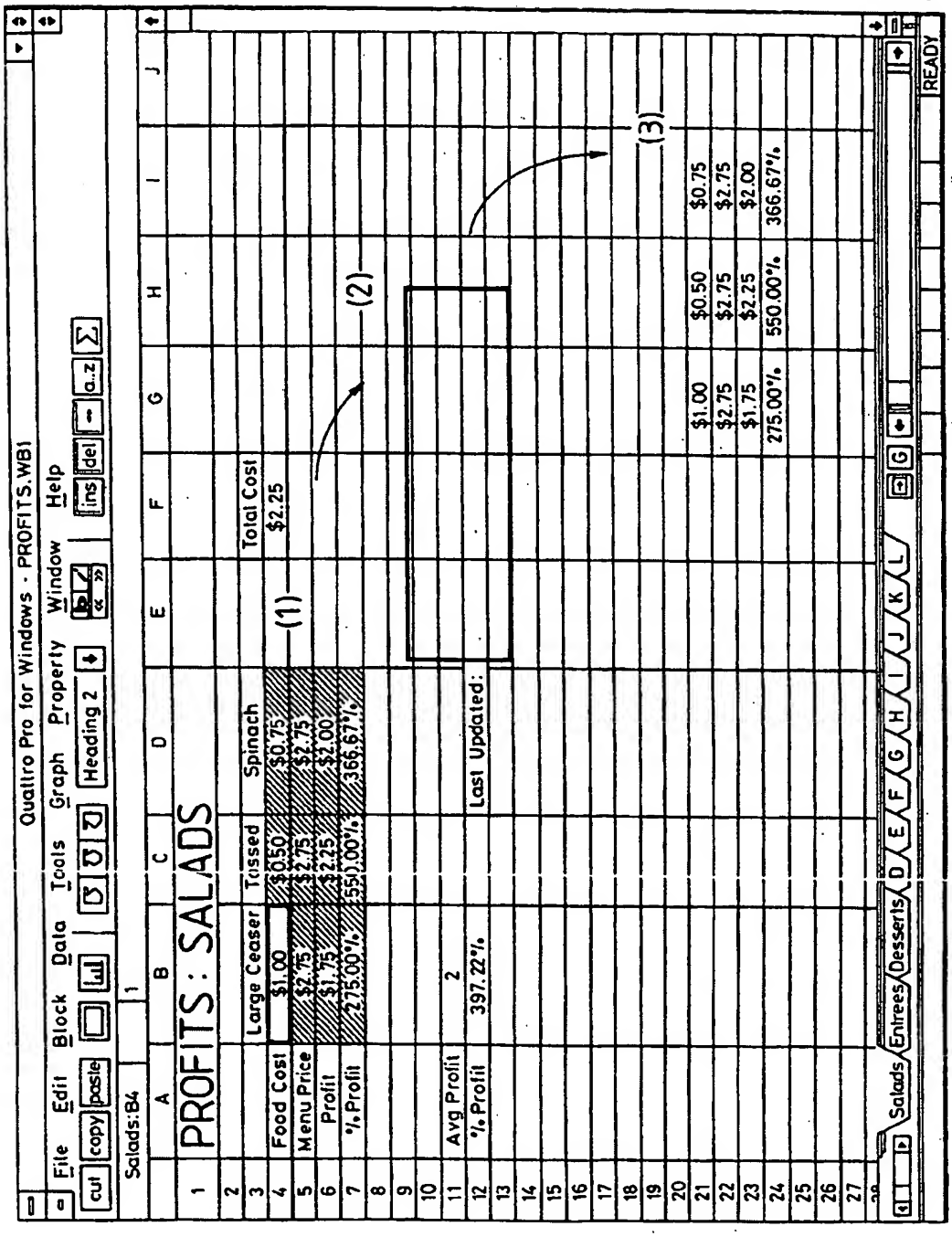


FIG. 4G

BNSDOCID: <EP___0569133A2_I_>

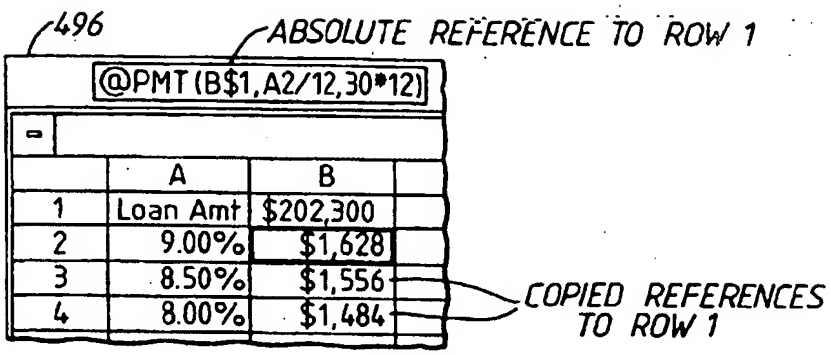


FIG.4H

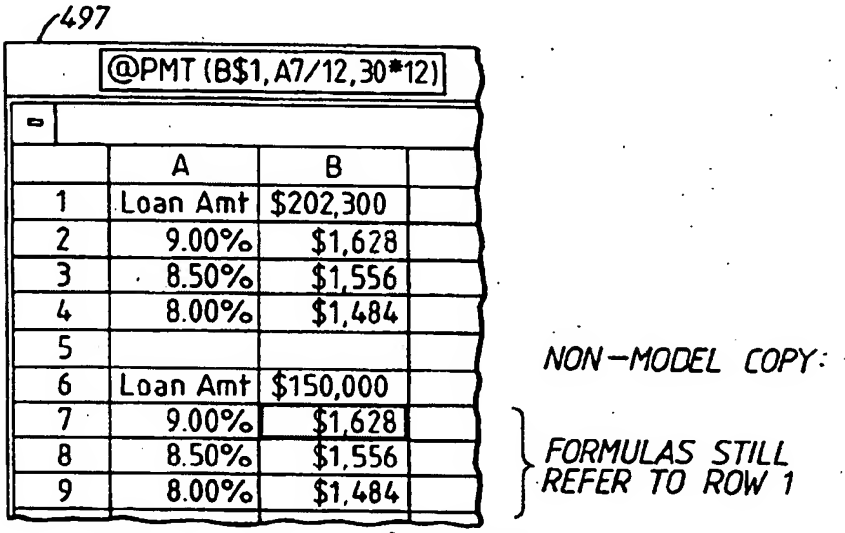


FIG.4I

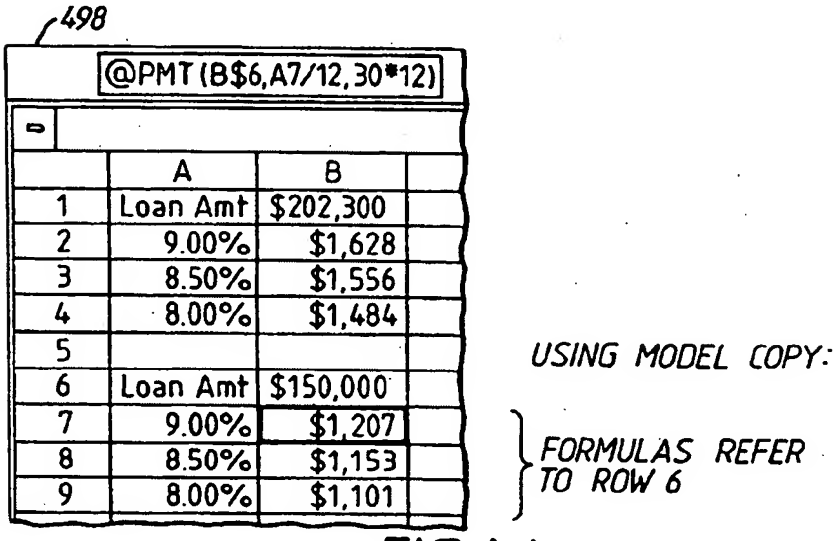


FIG.4J

BNSDOCID: <EP___0569133A2_I_>

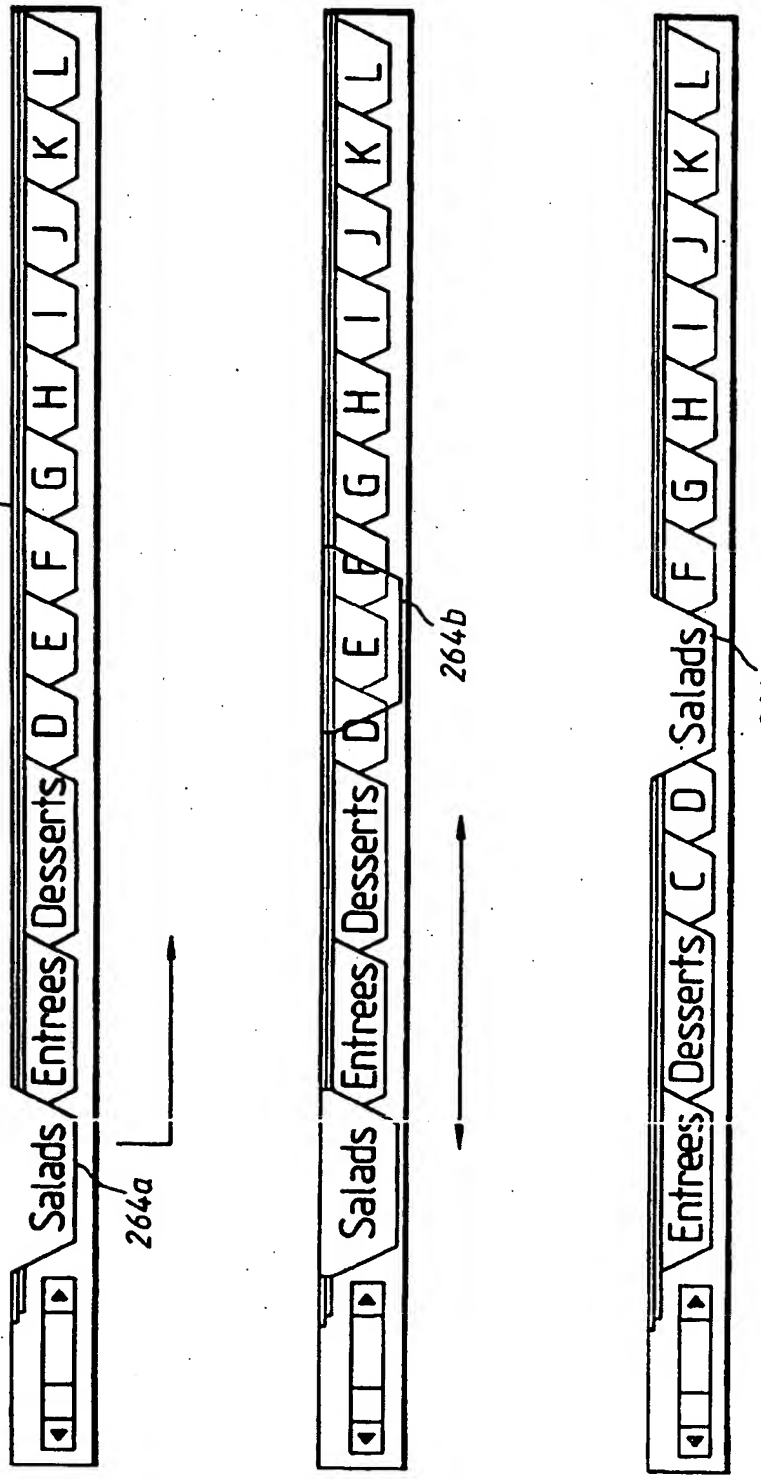
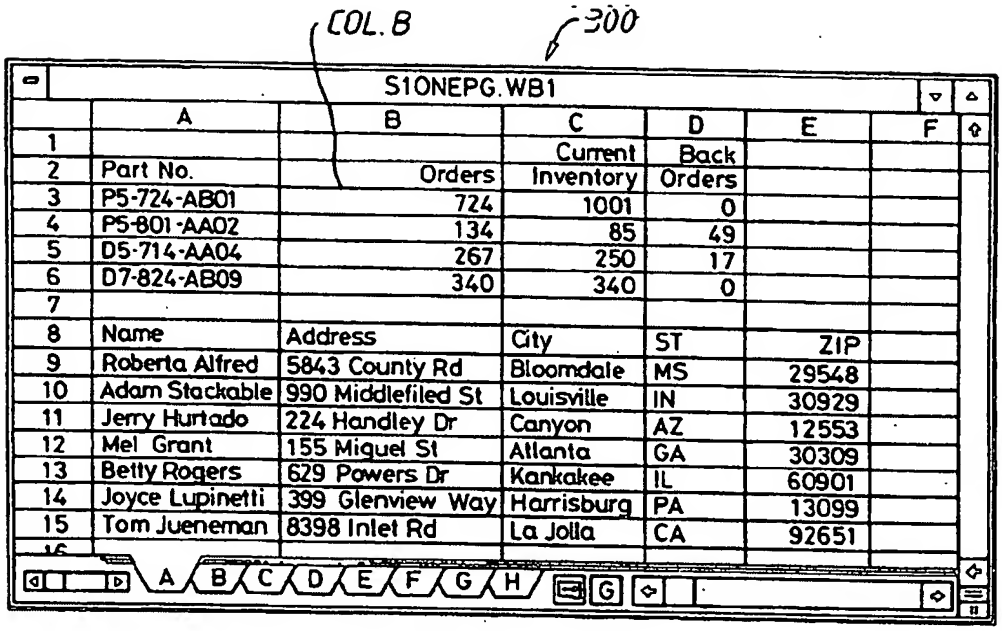


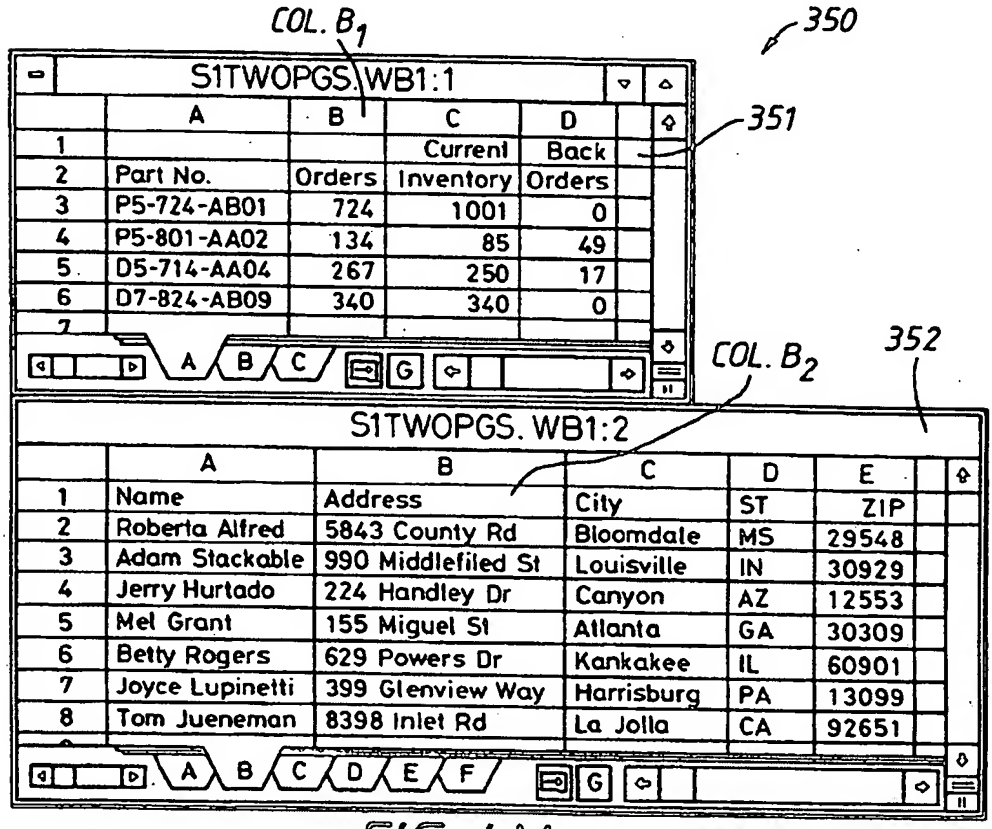
FIG. 4K

BNSDOCID: <EP___0569133A2_I_>



	A	B	C	D	E	F
1			Current	Back		
2	Part No.	Orders	Inventory	Orders		
3	P5-724-AB01	724	1001	0		
4	P5-801-AA02	134	85	49		
5	D5-714-AA04	267	250	17		
6	D7-824-AB09	340	340	0		
7						
8	Name	Address	City	ST	ZIP	
9	Roberta Alfred	5843 County Rd	Bloomdale	MS	29548	
10	Adam Stackable	990 Middlefiled St	Louisville	IN	30929	
11	Jerry Hurtado	224 Handley Dr	Canyon	AZ	12553	
12	Mel Grant	155 Miguel St	Atlanta	GA	30309	
13	Betty Rogers	629 Powers Dr	Kankakee	IL	60901	
14	Joyce Lupinetti	399 Glenview Way	Harrisburg	PA	13099	
15	Tom Jueneman	8398 Inlet Rd	La Jolla	CA	92651	

FIG. 4L



	A	B	C	D
1			Current	Back
2	Part No.	Orders	Inventory	Orders
3	P5-724-AB01	724	1001	0
4	P5-801-AA02	134	85	49
5	D5-714-AA04	267	250	17
6	D7-824-AB09	340	340	0
7				

	A	B	C	D	E
1	Name	Address	City	ST	ZIP
2	Roberta Alfred	5843 County Rd	Bloomdale	MS	29548
3	Adam Stackable	990 Middlefiled St	Louisville	IN	30929
4	Jerry Hurtado	224 Handley Dr	Canyon	AZ	12553
5	Mel Grant	155 Miguel St	Atlanta	GA	30309
6	Betty Rogers	629 Powers Dr	Kankakee	IL	60901
7	Joyce Lupinetti	399 Glenview Way	Harrisburg	PA	13099
8	Tom Jueneman	8398 Inlet Rd	La Jolla	CA	92651

FIG. 4M

BNSDOCID: <EP___0569133A2_1_>

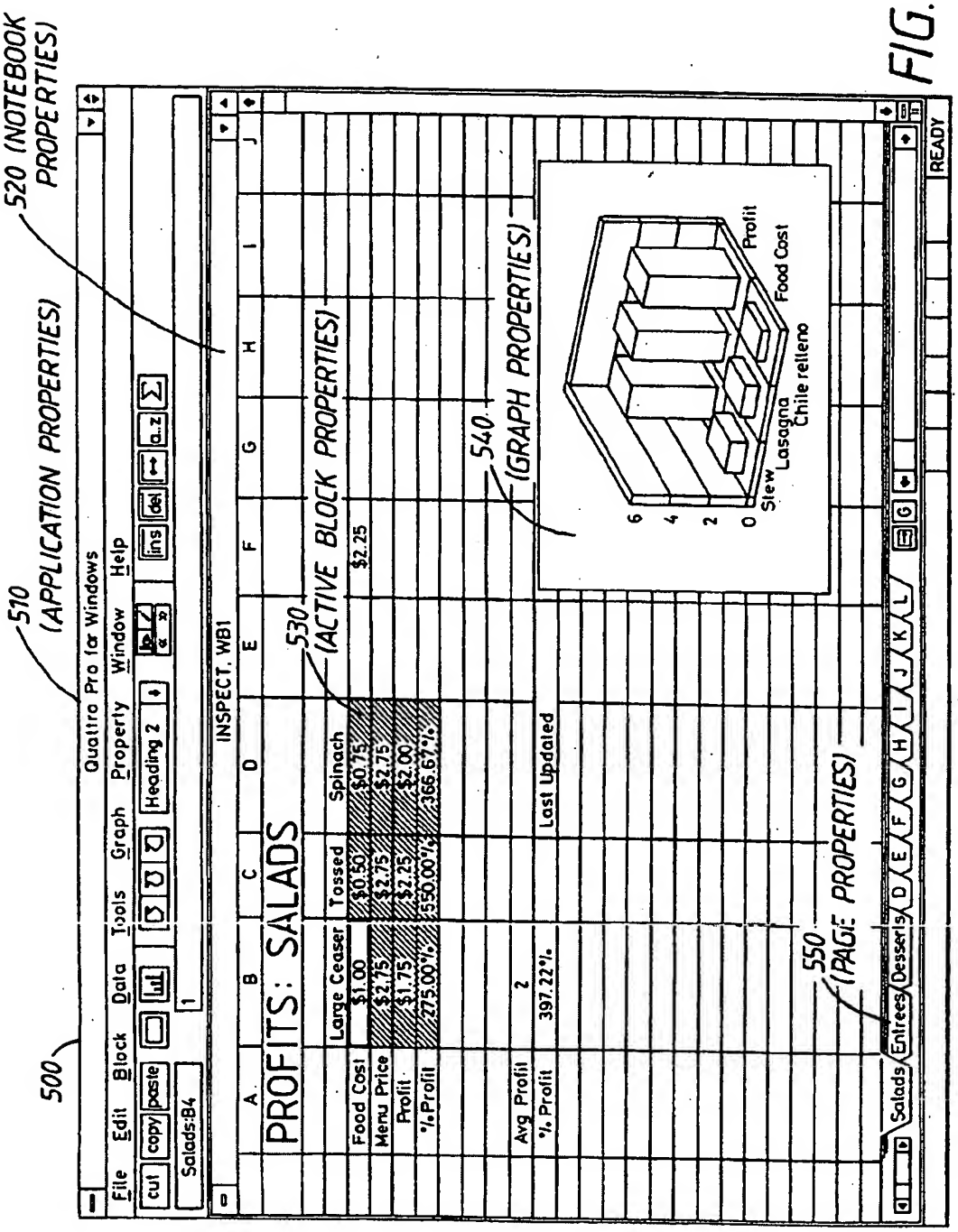


FIG.5A

BNSDOCID: <EP___0569133A2_1_>

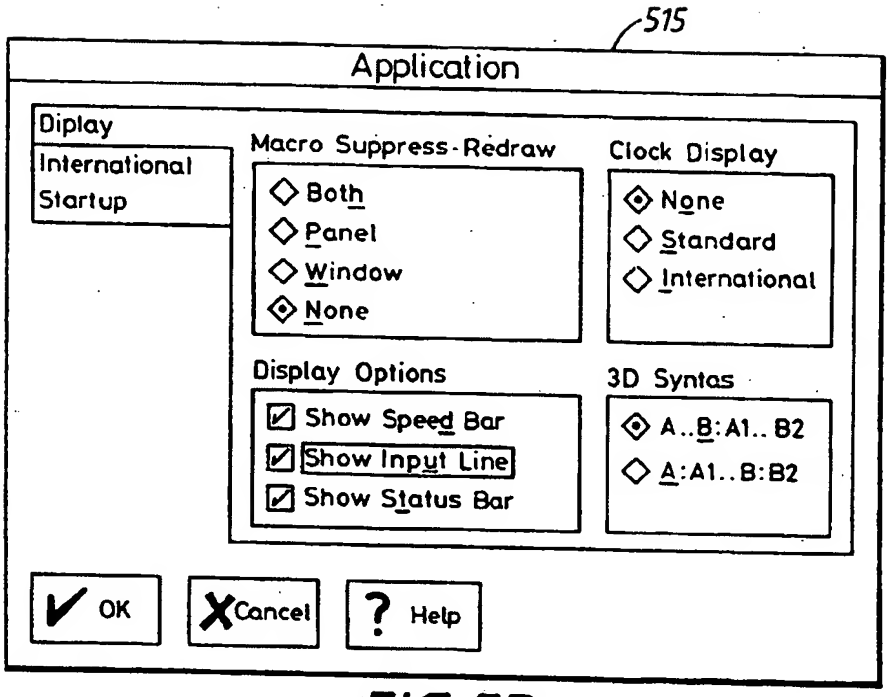


FIG. 5B

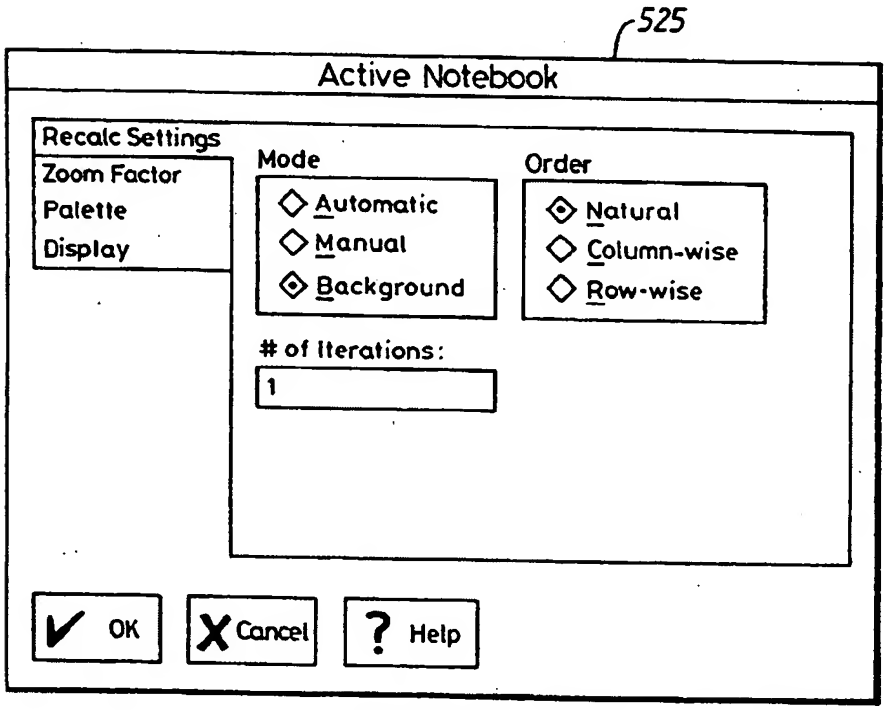


Fig. 5C

BNSDOCID: <EP___0569133A2_I_>

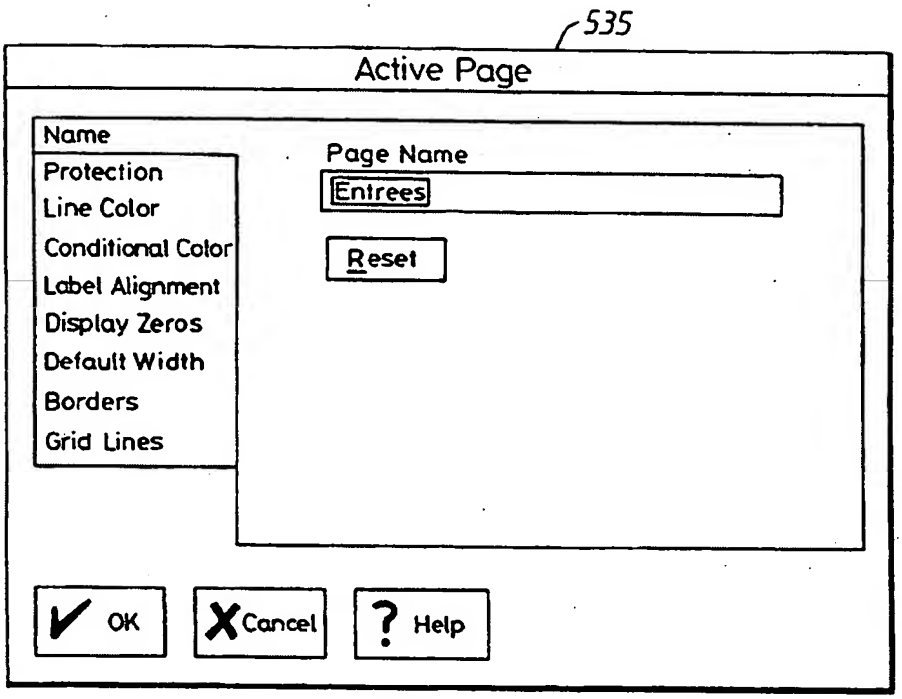


FIG. 5D

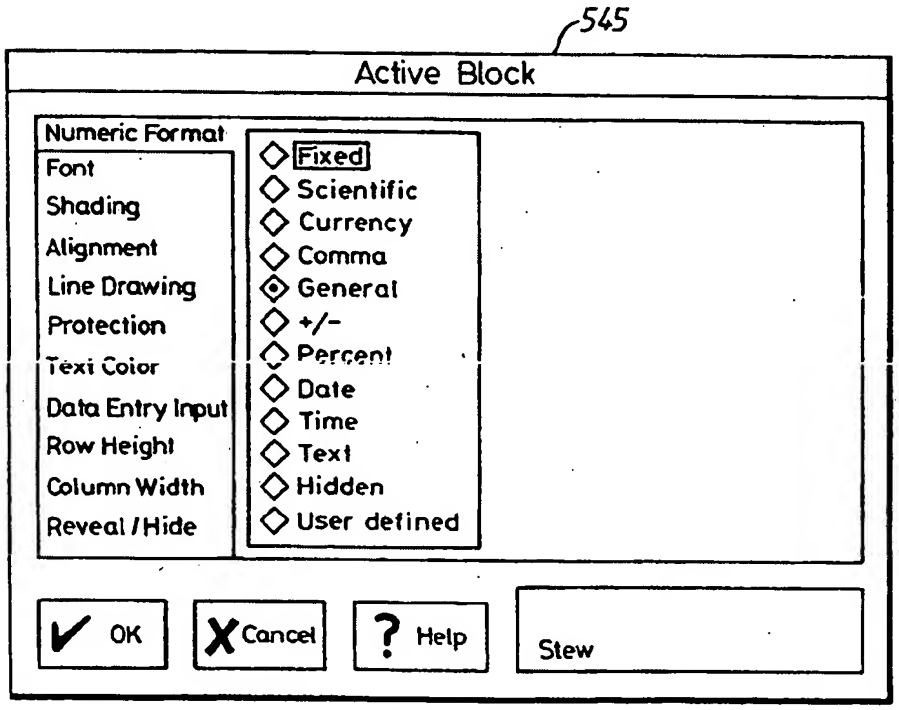


FIG. 5E

BNSDOCID: <EP__0569133A2_I_>

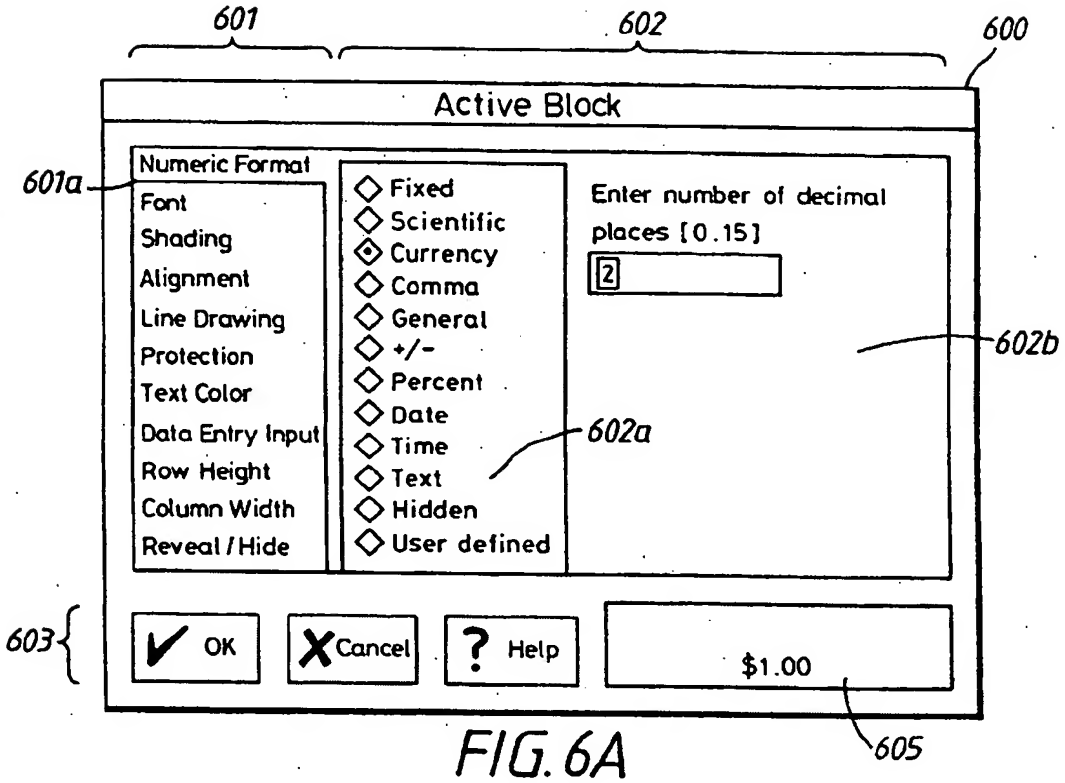


FIG. 6A

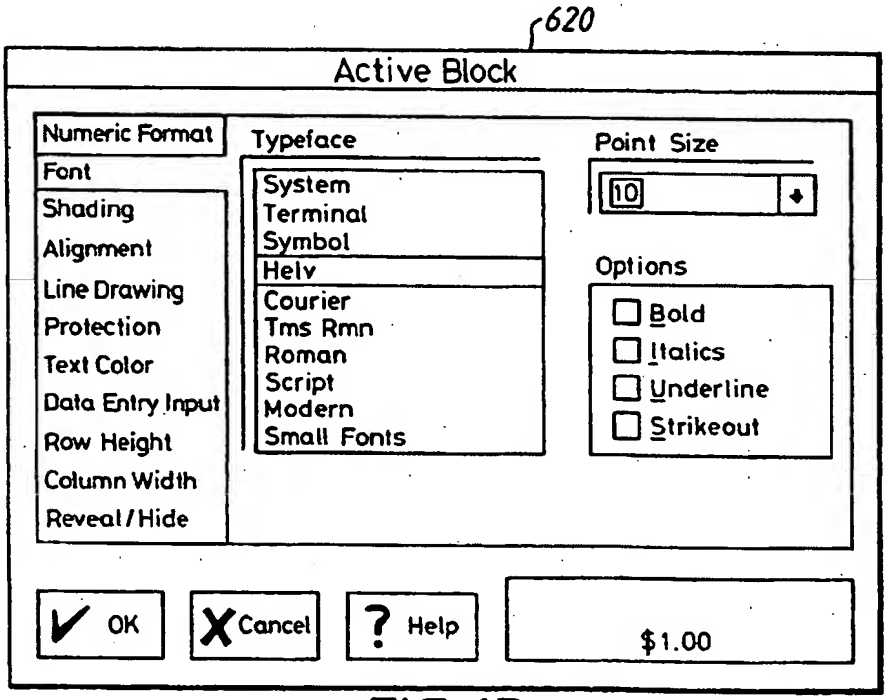


FIG. 6B

BNSDOCID: <EP___0569133A2_I_>

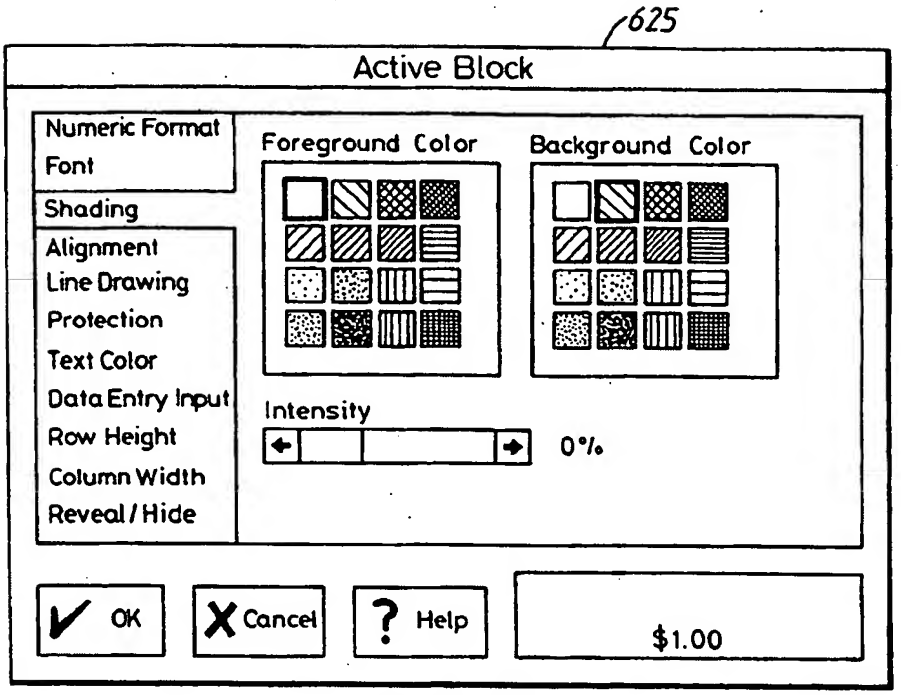


FIG. 6C

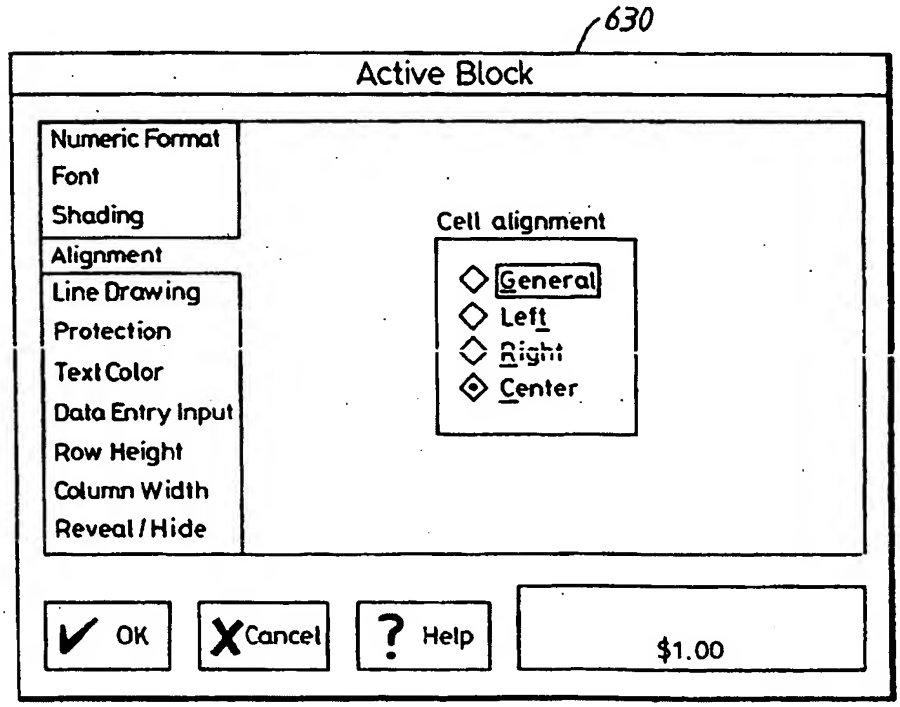


FIG. 6D

BNSDOCID: <EP___0569133A2_I_>

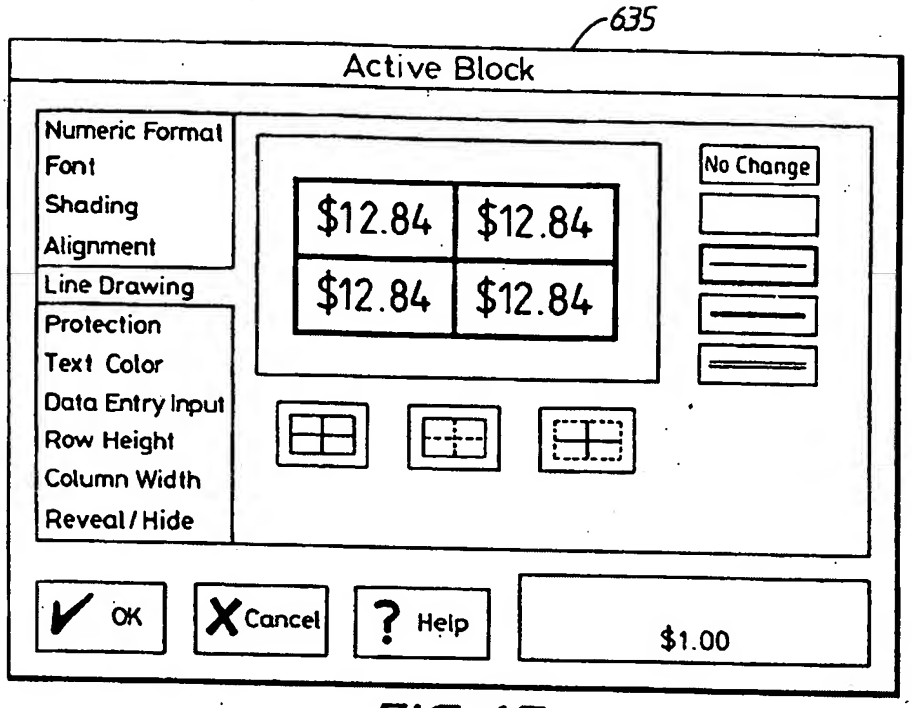


FIG. 6E

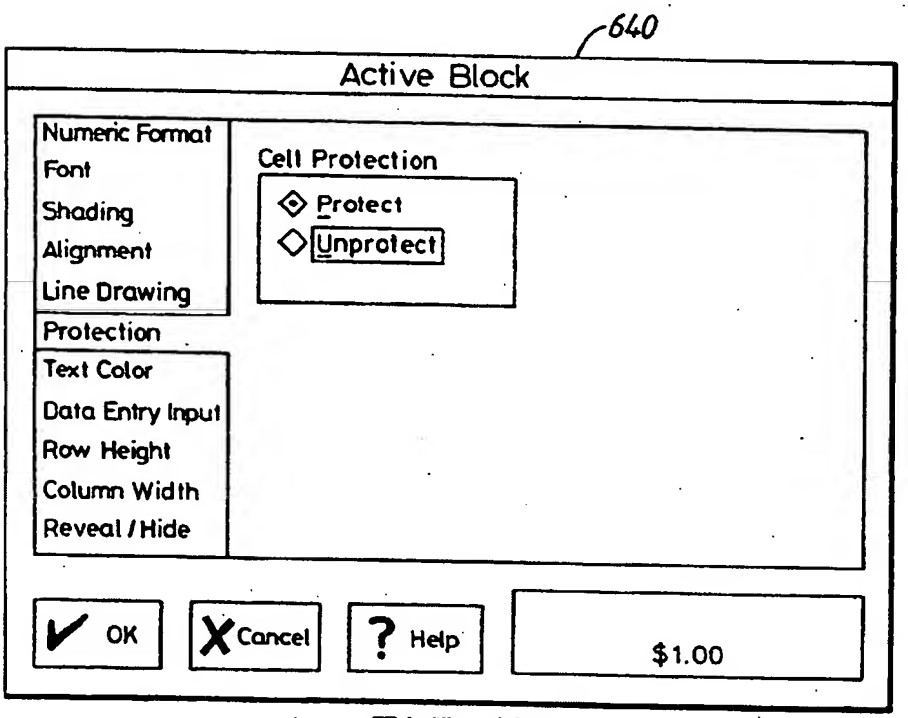


FIG. 6F

BNSDOCID: <EP___0569133A2_I_>

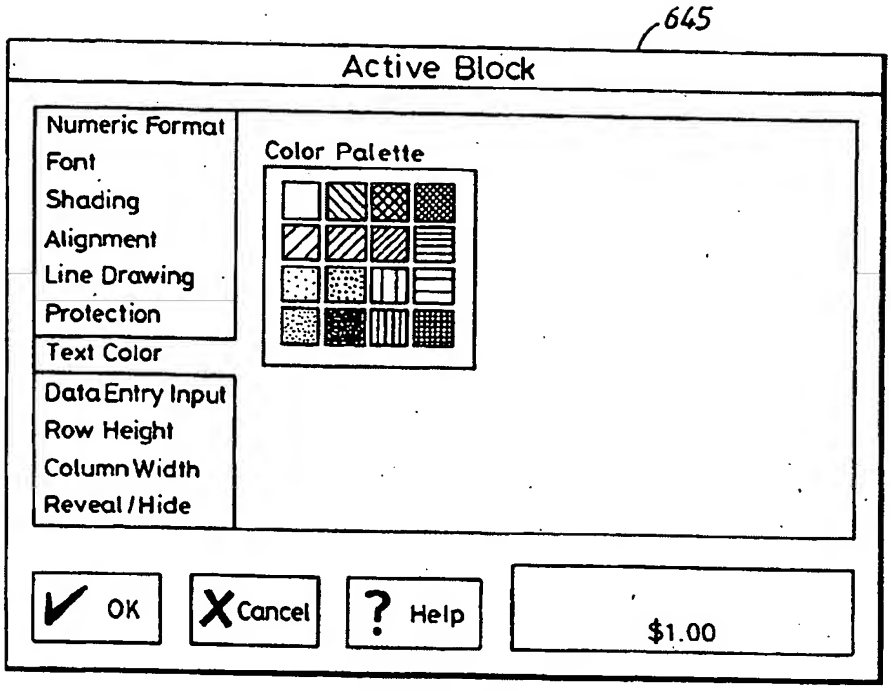


FIG. 6G

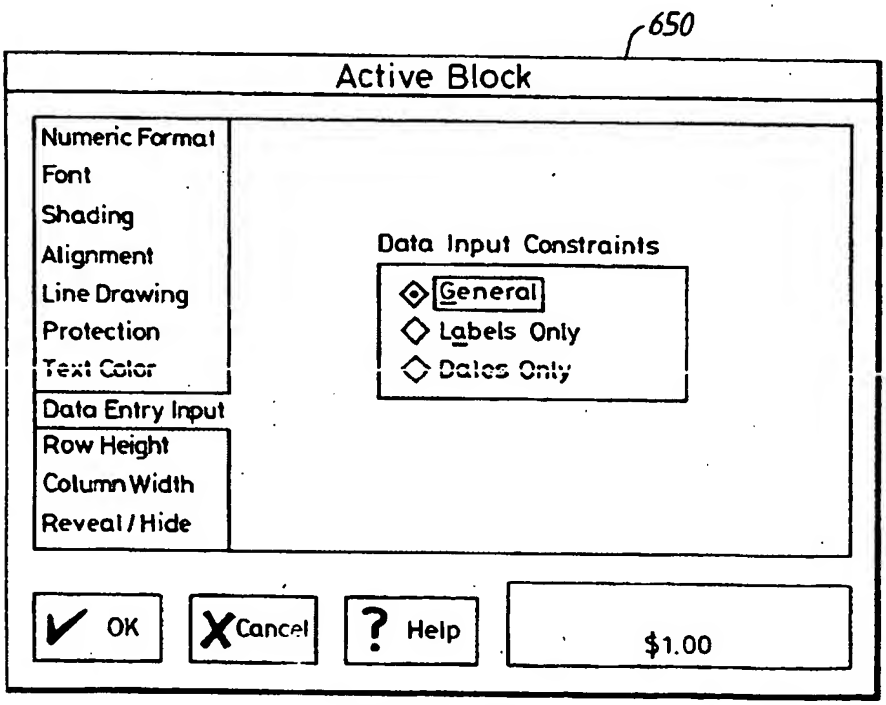


FIG. 6H

BNSDOCID: <EP___0569133A2_I_>

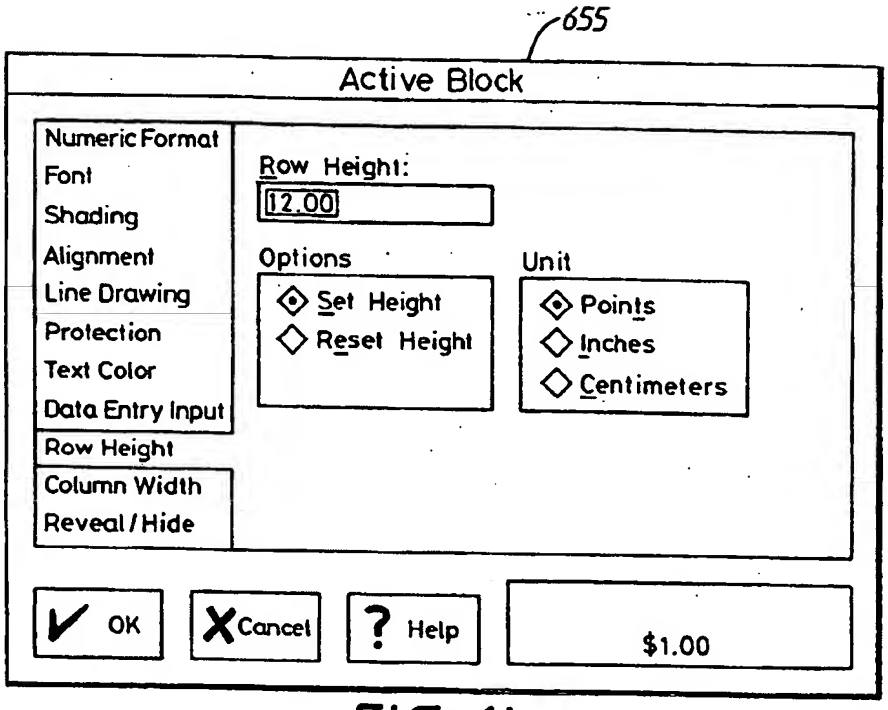


FIG. 6I

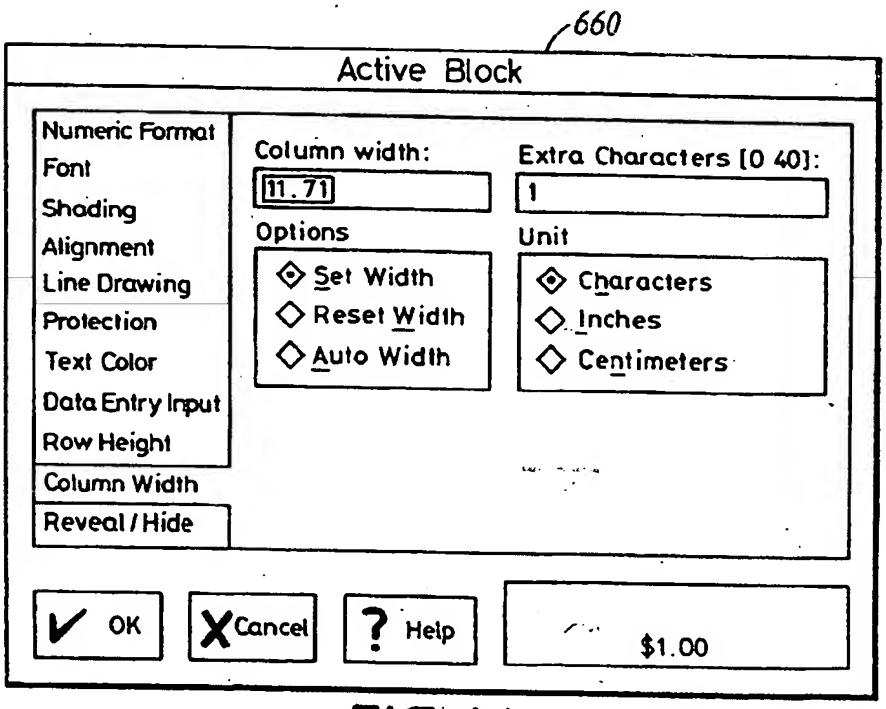


FIG. 6J

BNSDOCID: <EP___0569133A2_I_>

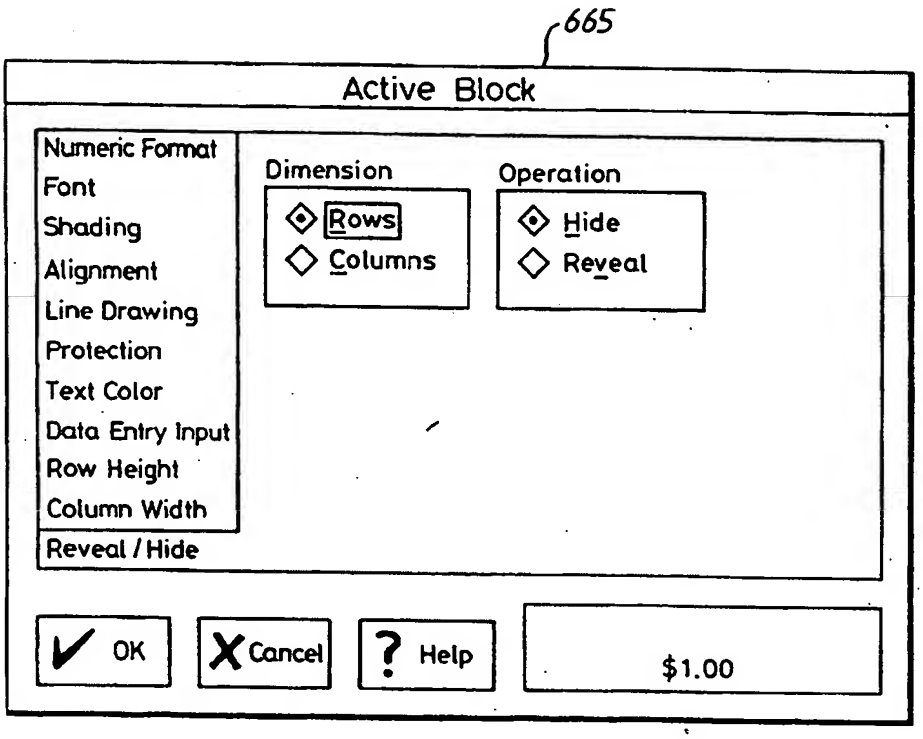


FIG. 6K

BNSDOCID: <EP___0569133A2_I_>

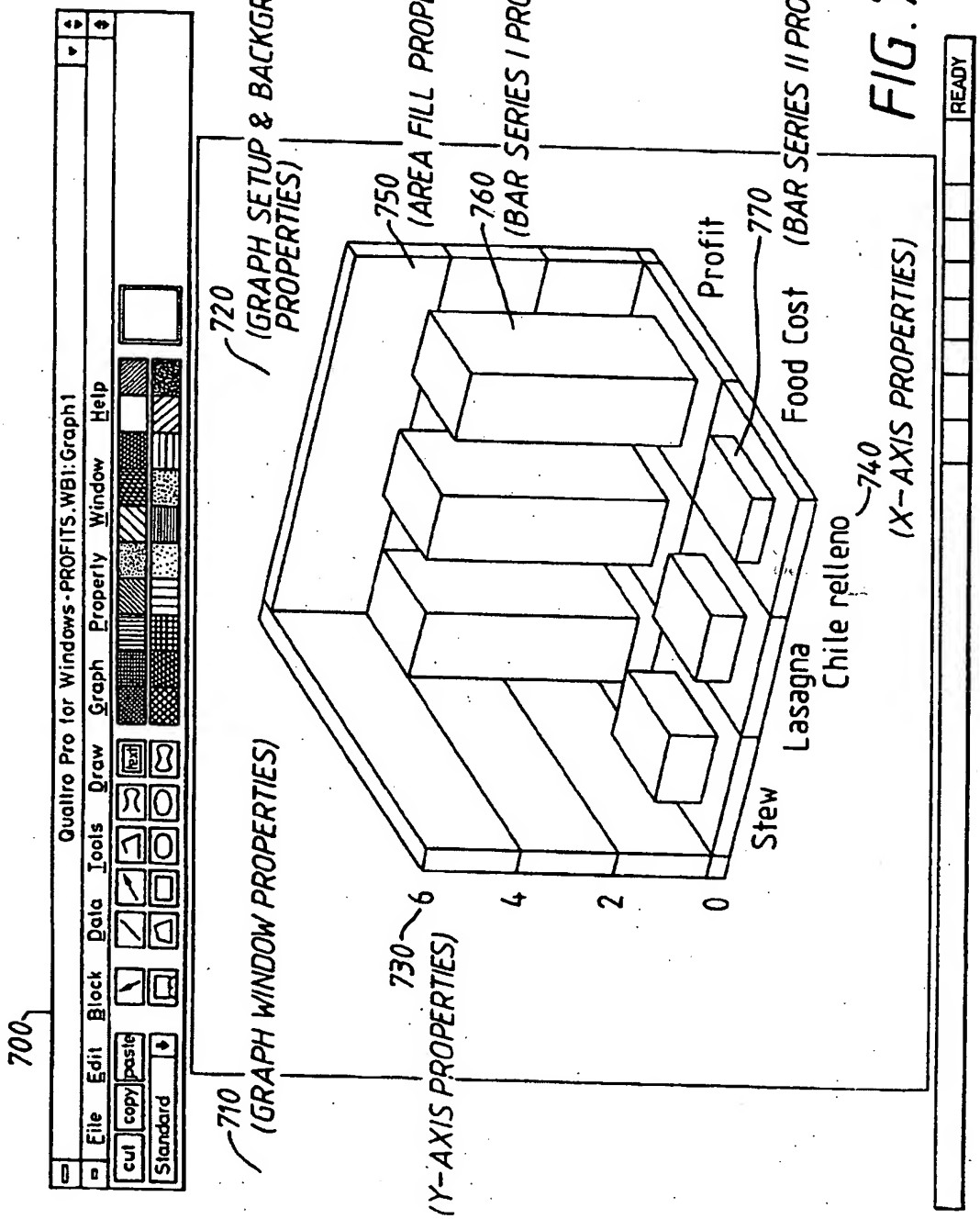


FIG. 7A

BNSDOCID: <EP___0569133A2_I_>

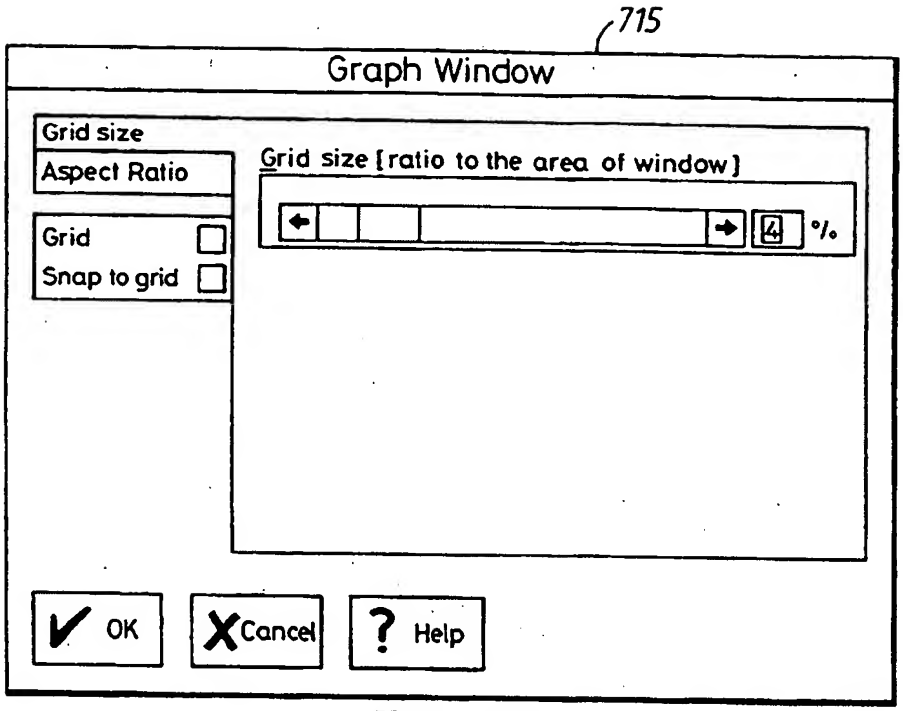


FIG. 7B

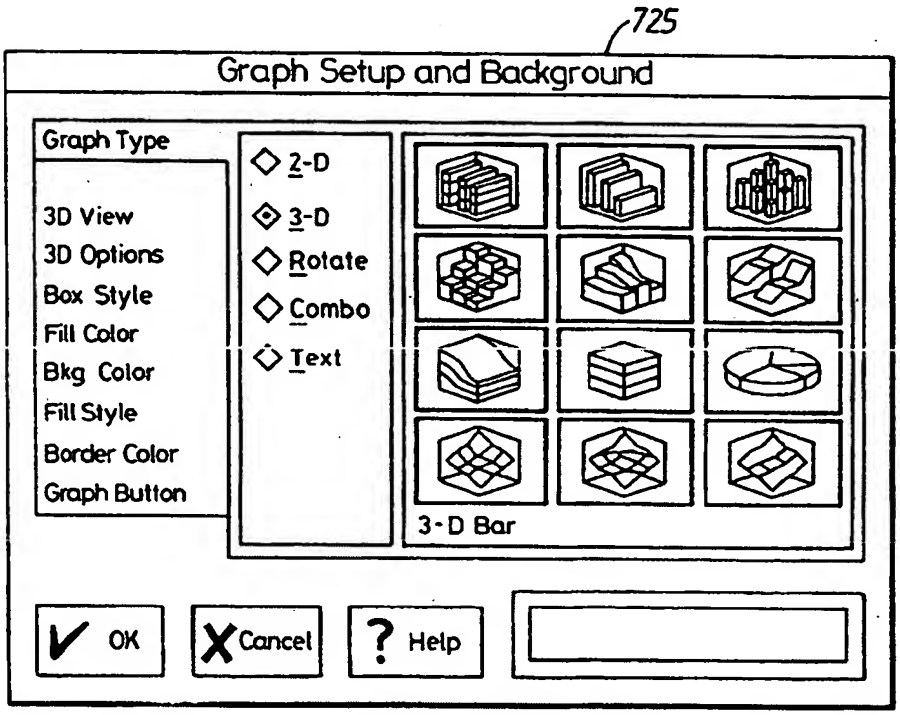


FIG. 7C

BNSDOCID: <EP___0569133A2_I_>

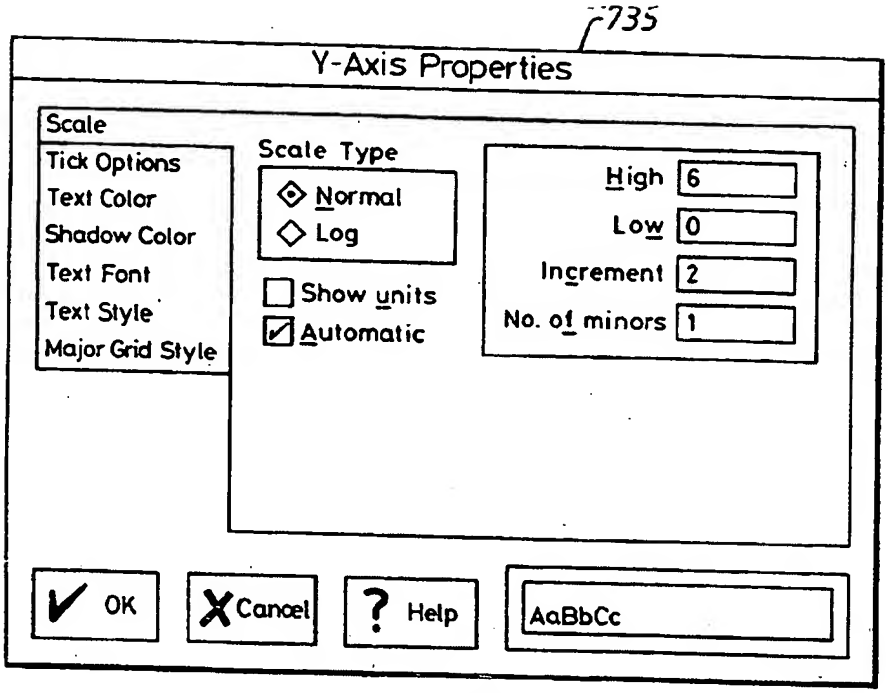


FIG. 7D

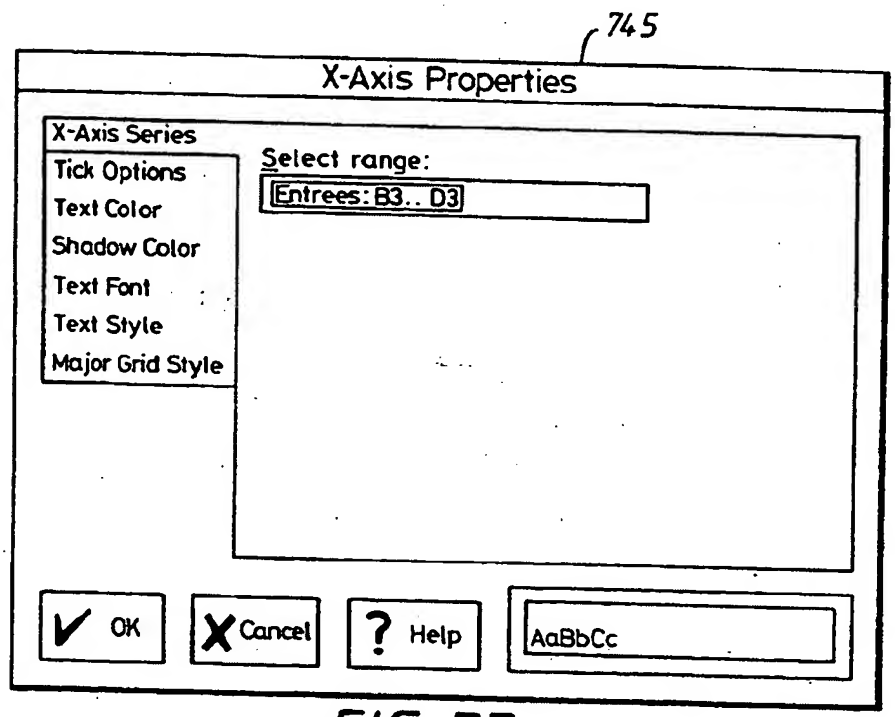


FIG. 7E

BNSDOCID: <EP___0569133A2_I_>

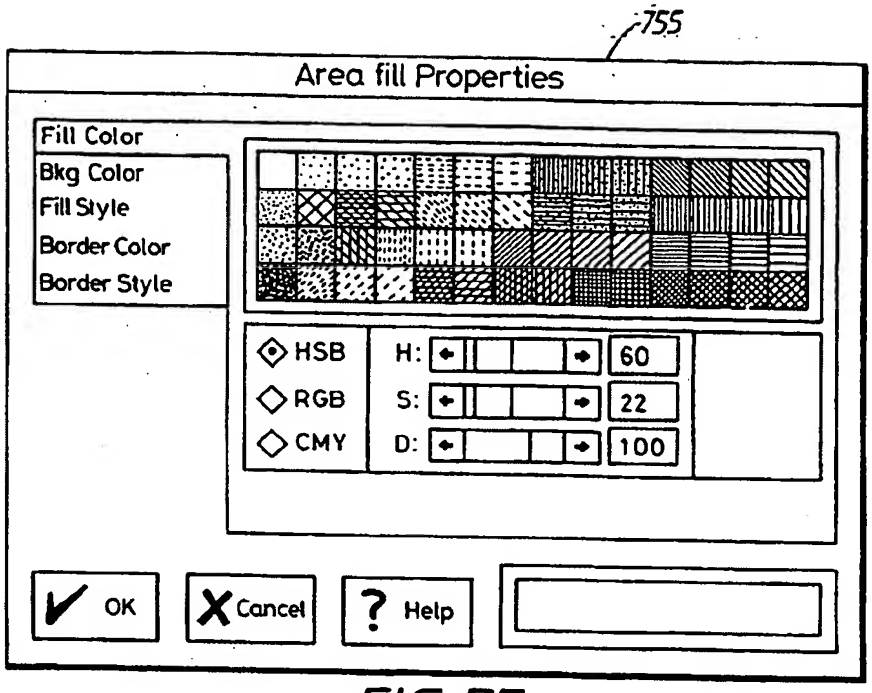


FIG. 7F

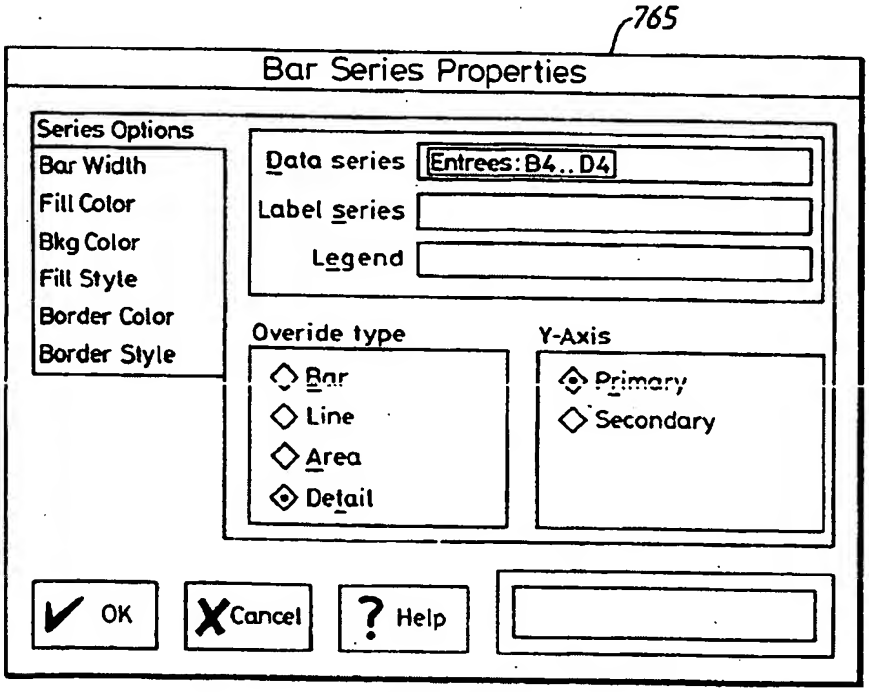


FIG. 7G

BNSDOCID: <EP___0569133A2_I_>

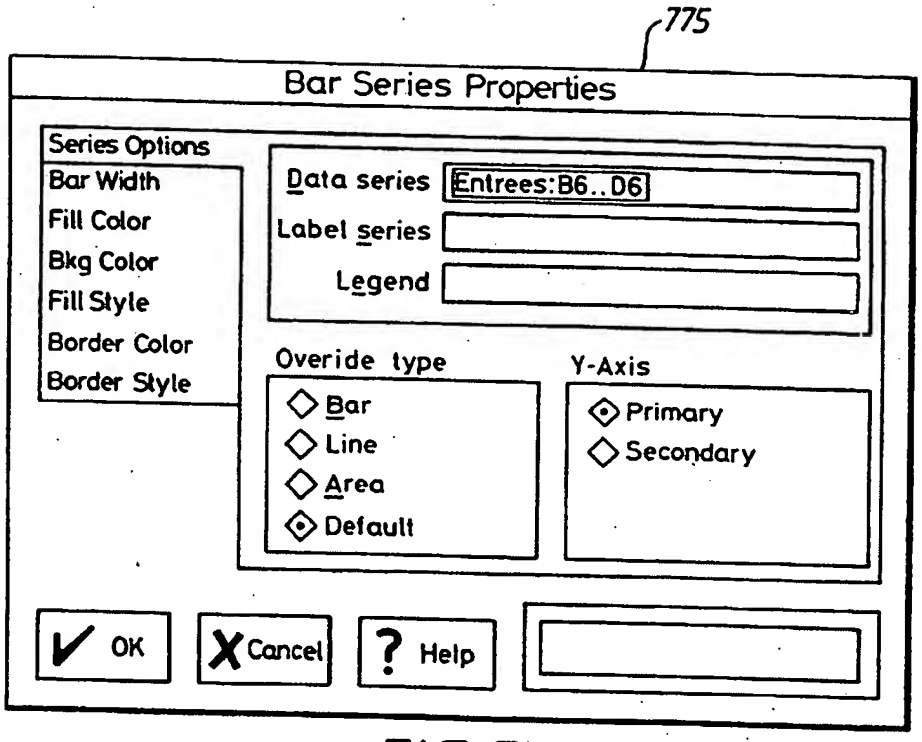


FIG. 7H

BNSDOCID: <EP___0569133A2_I_>

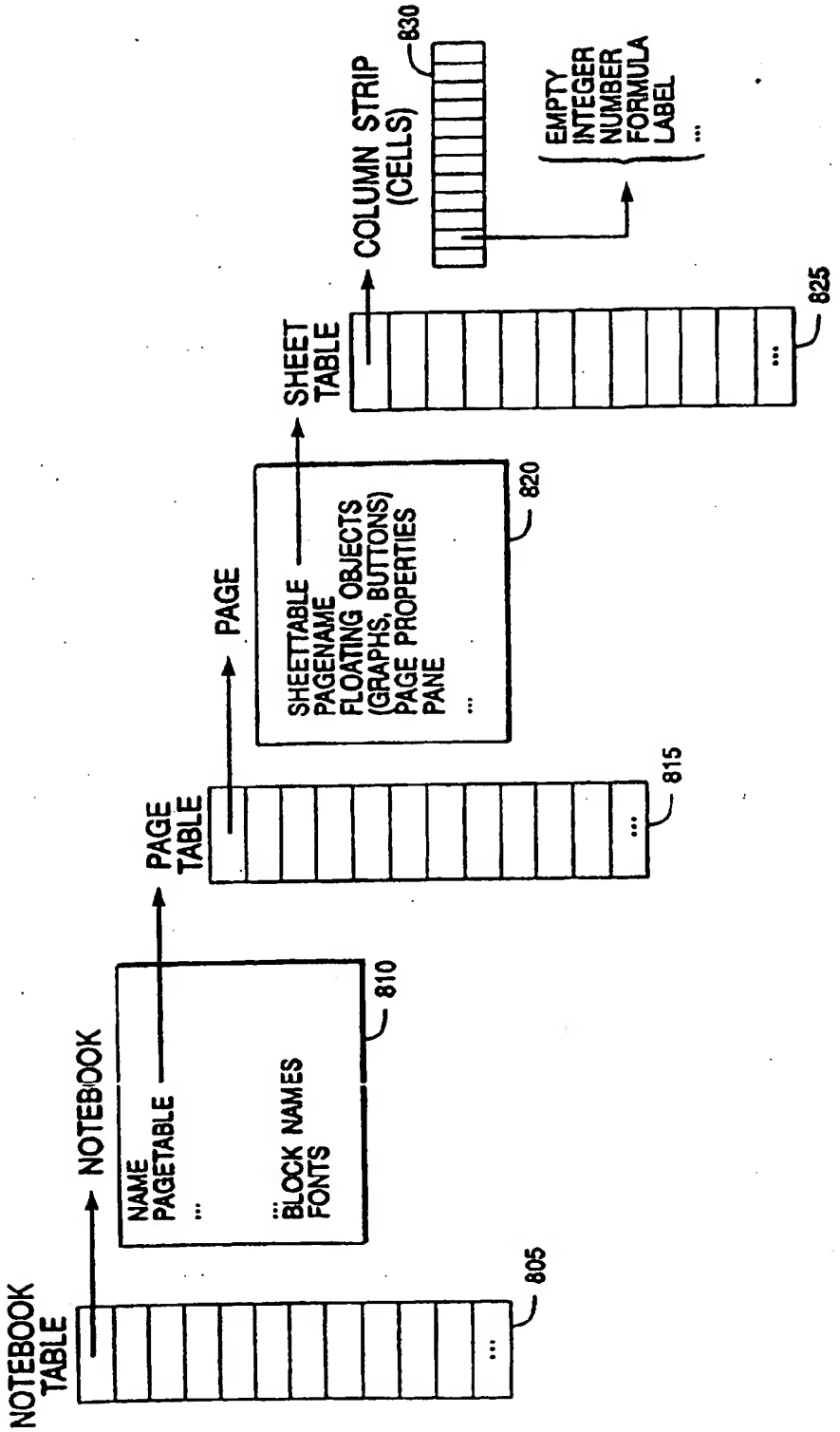


FIG. 8A

BNSDOCID: <EP___0569133A2_I_>

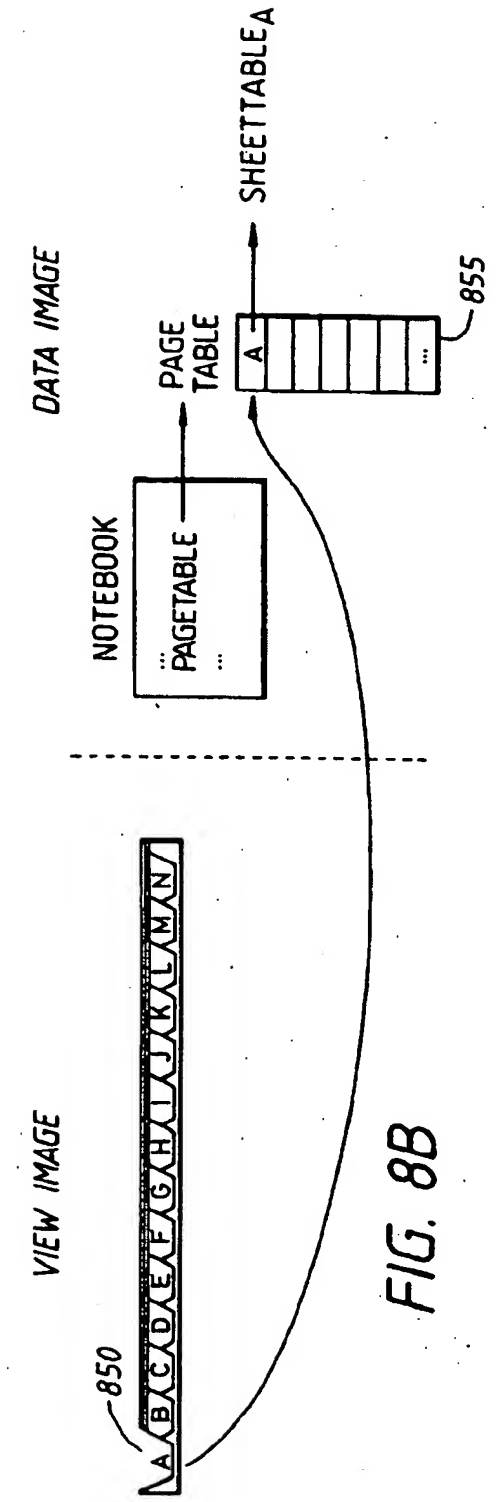


FIG. 8B

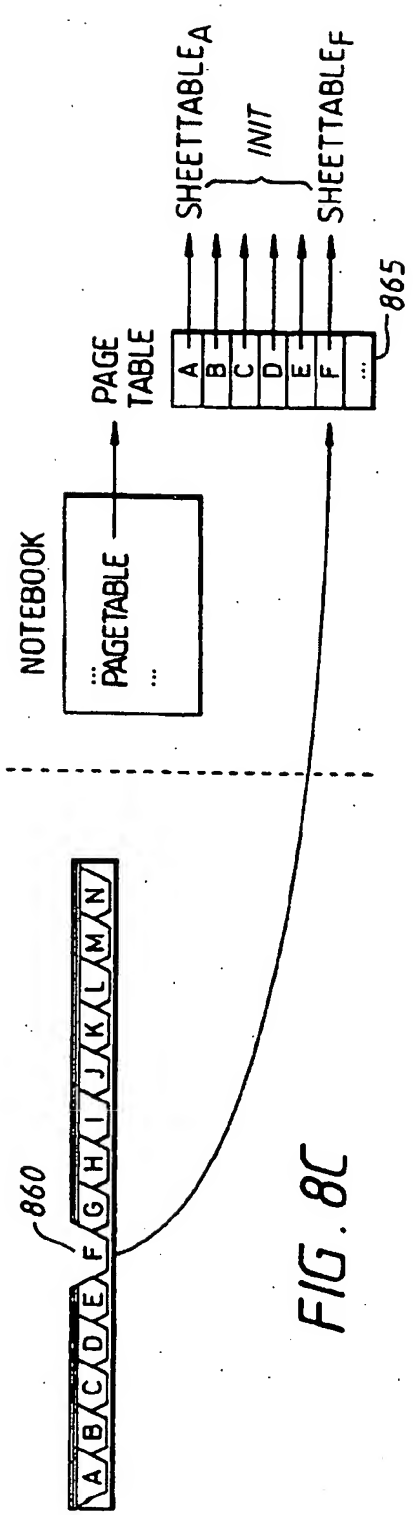


FIG. 8C

BNSDOCID: <EP___0569133A2_I_>

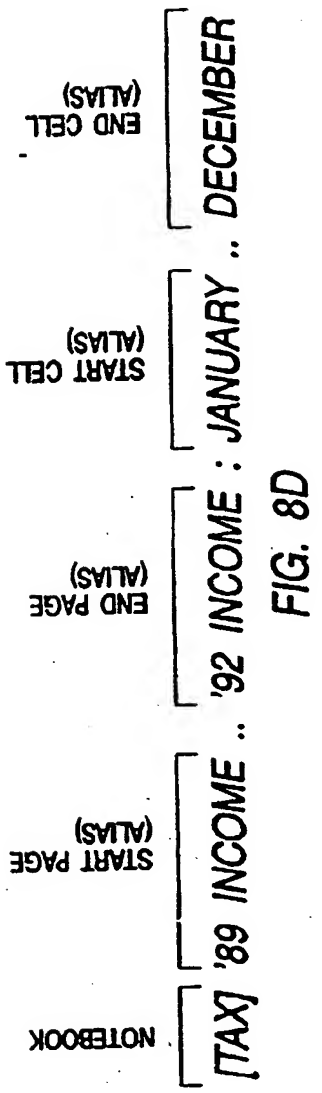


FIG. 8D

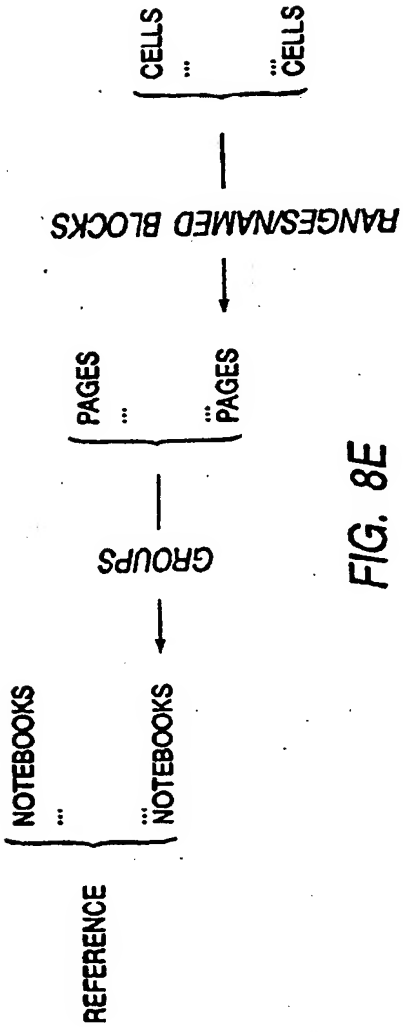


FIG. 8E

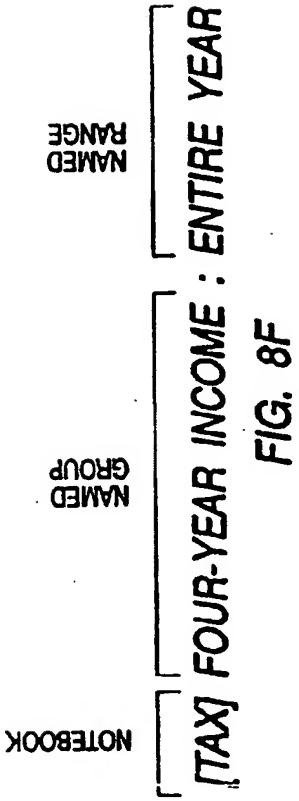


FIG. 8F

BNSDOCID: <EP___0569133A2_I_>

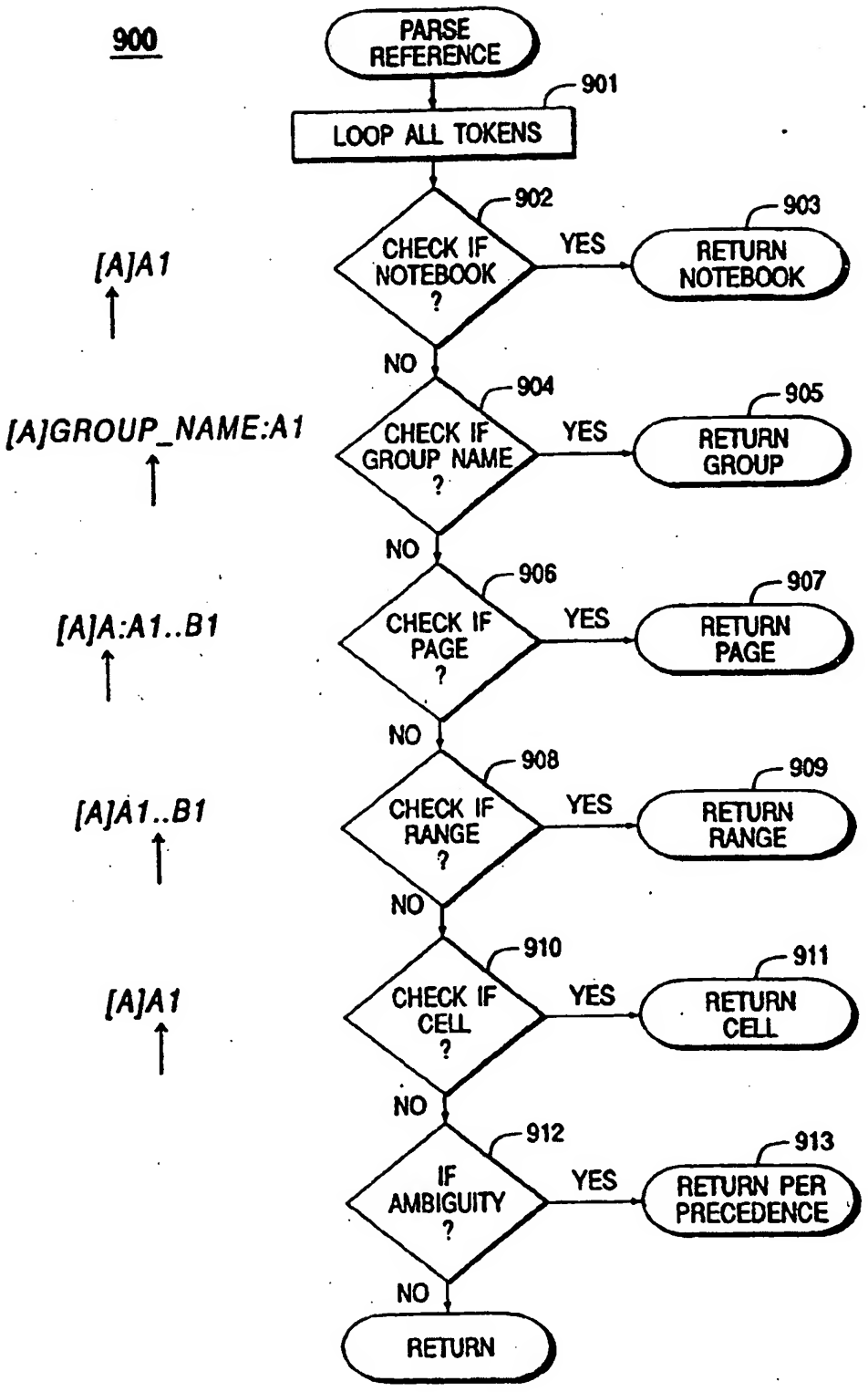


FIG. 9A

BNSDOCID: EP _0569133A2_I_>

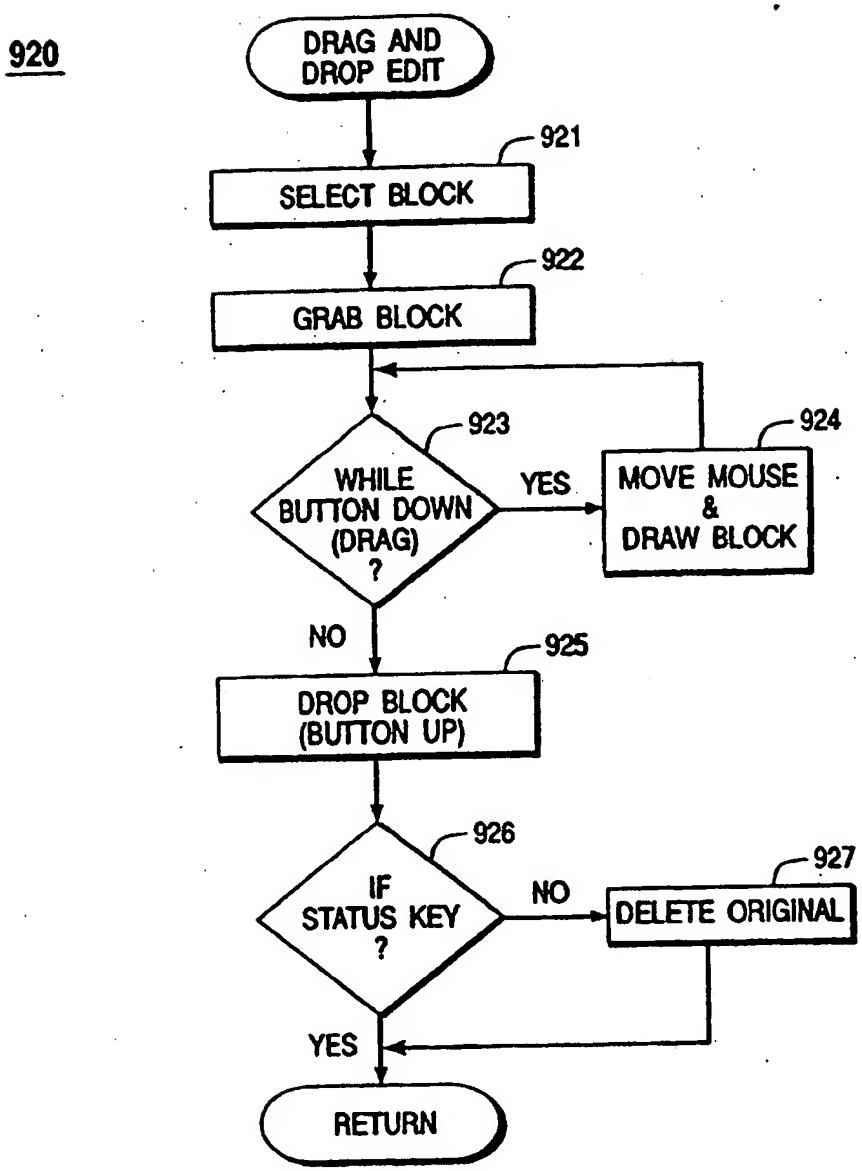


FIG. 9B

BNSDOCID: <EP___0569133A2_I_>

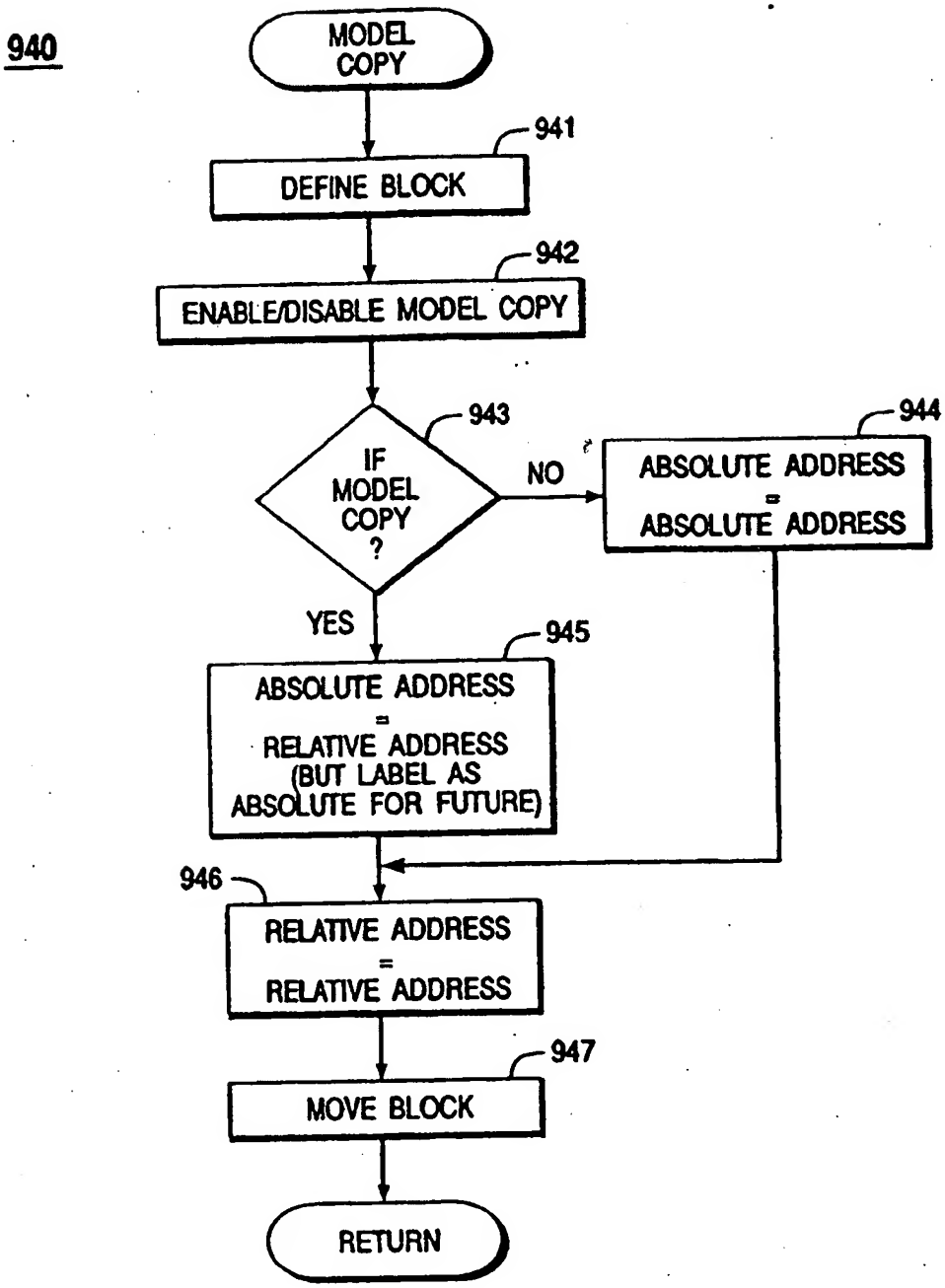


FIG. 9C

BNSDOCID: <EP__0569133A2_I_>

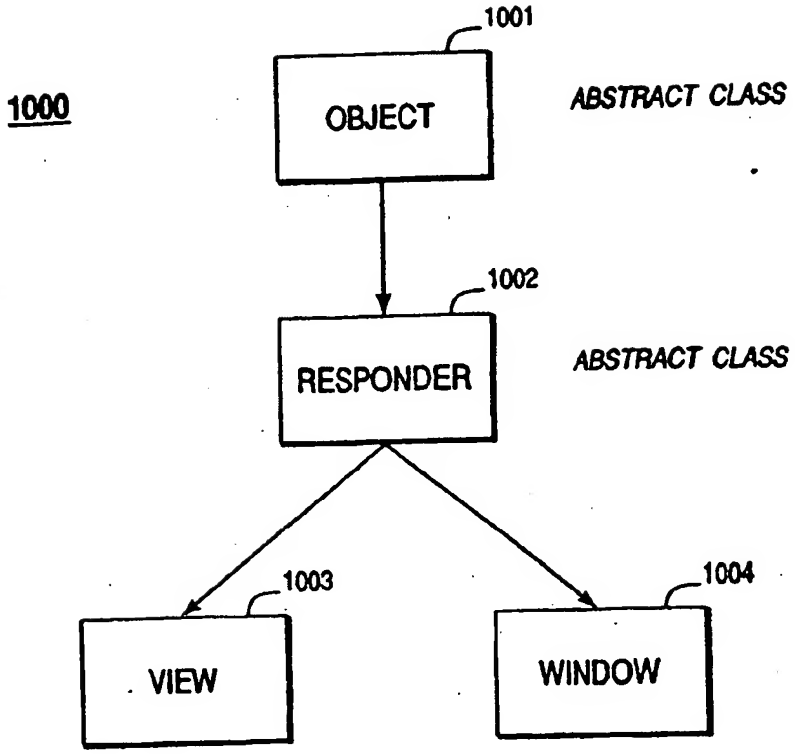


FIG. 10A

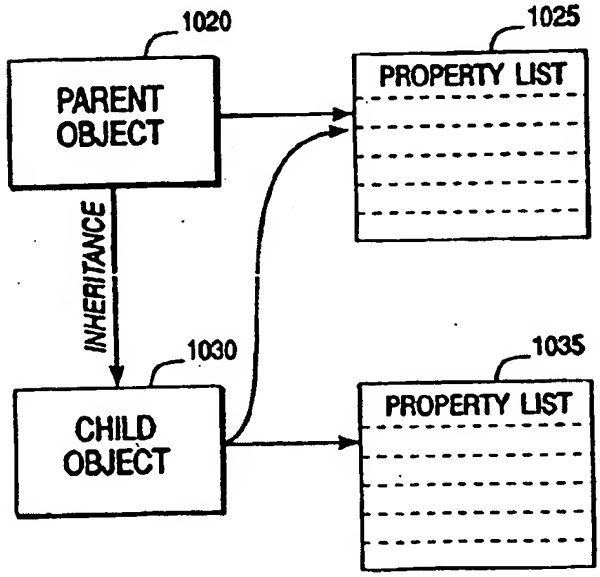


FIG. 10B

BNSDOCID: <EP___0569133A2_I_>

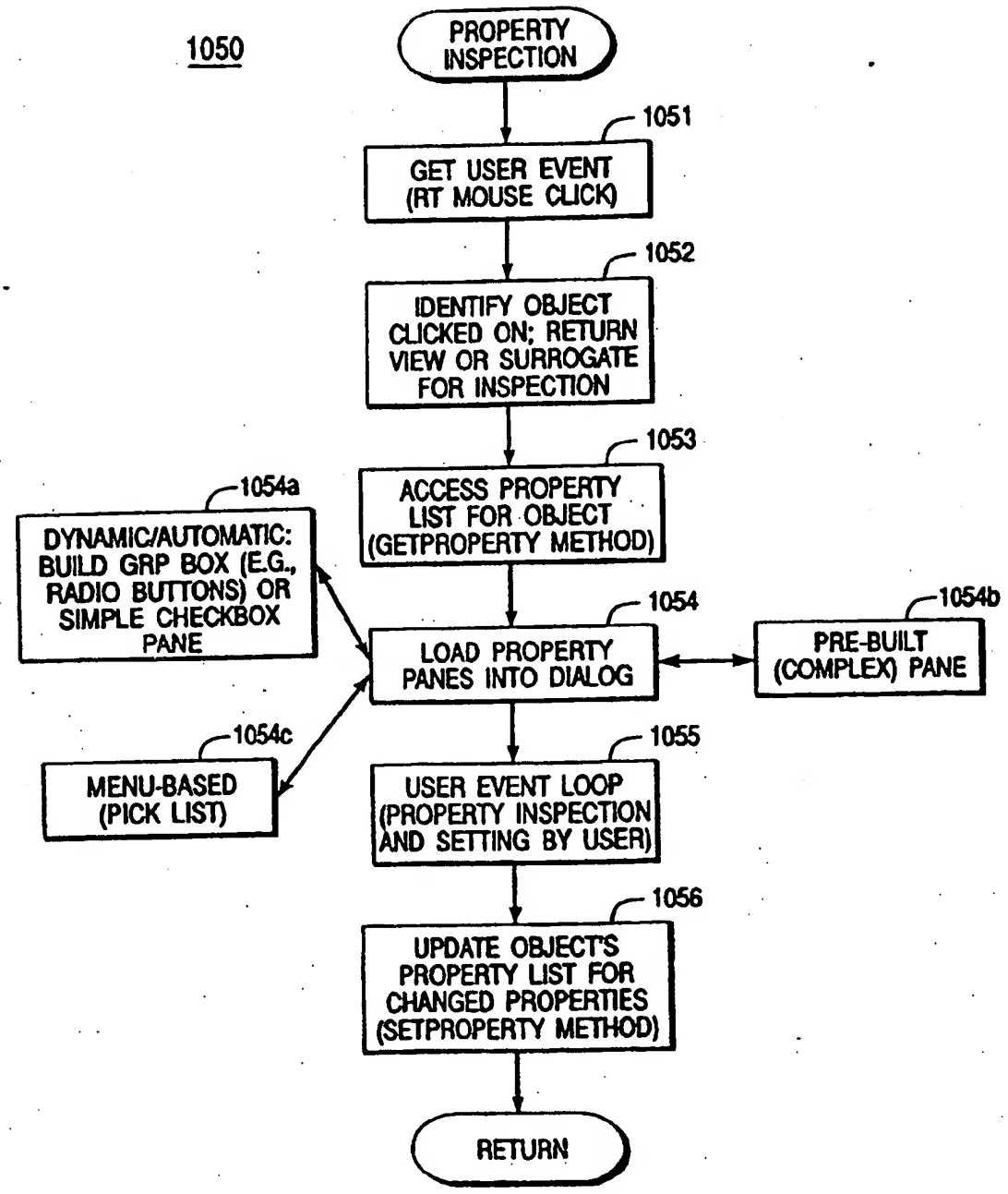


FIG. 10C

BNSDOCID: <EP___0569133A2_I_>

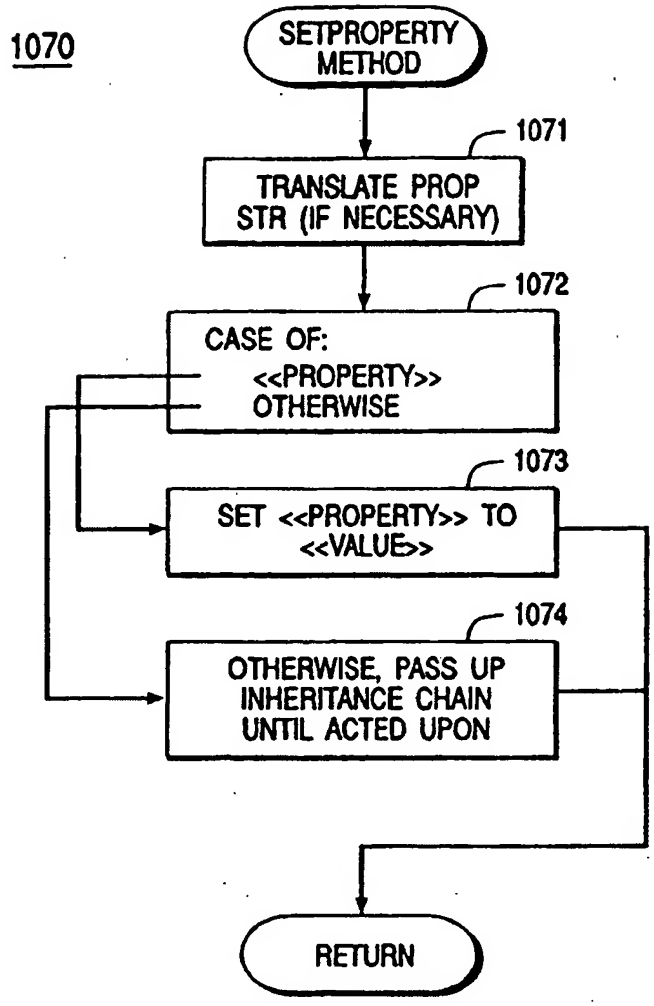


FIG. 10D

BNSDOCID: <EP___0569133A2_I_>

(19) Europäisches Patentamt
European Patent Offic
Offic européen d s brev ts

(11) Publication number : **0 569 133 A3**

(12) EUROPEAN PATENT APPLICATION

(21) Application number : **93302717.9**

(22) Date of filing : **07.04.93**

(51) Int. Cl.[5] : **G06F 15/20**, G06F 9/44, G06F 3/033

(30) Priority : **08.04.92 US 866658**

(43) Date of publication of application : **10.11.93 Bulletin 93/45**

(84) Designated Contracting States : **AT BE CH DE DK ES FR GB GR IE IT LI LU MC NL PT SE**

(88) Date of deferred publication of search report : **09.02.94 Bulletin 94/06**

(71) Applicant : **BORLAND INTERNATIONAL, Inc.**
1800 Green Hills Road
Scott's Valley, California 95067 (US)

(72) Inventor : **Anderson, Charles R.**
323 Arroyo Seco
Santa Cruz, California 95060 (US)

Inventor : **Warfield, Robert W.**
3470 Merrill Street
Aptos, California 95003 (US)
Inventor : **Cseri, Istvan**
15 Terrace View Court
Scotts Valley, California 95066 (US)
Inventor : **Low, Murray K.**
802 Columbia Street
Santa Cruz, California 95060 (US)
Inventor : **Liaw, Weikuo**
104 Nanna Court
Santa Cruz, California 95060 (US)
Inventor : **Bush, Alan M.**
905 Fremont Place, No.4
Menlo Park, California 94025 (US)

(74) Representative : **Godsill, John Kenneth et al**
Haseltine Lake & Co. Hazlitt House 28
Southampton Buildings Chancery Lane
London WC2A 1AT (GB)

(54) System and methods for computer interfaces.

(57) An electronic interface to a spreadsheet system includes a notebook interface (200) having a plurality of notebook pages (250), each of which may contain a spread of information cells, or other desired page type (e.g., Graphs page 700). Methods (900, 920, 940, 1050) are provided for rapidly accessing and processing information on the different pages, including displaying a plurality of page identifiers for selecting individual pages, and further including a preferred syntax for referencing information. Additional methods are provided for editing cells and blocks of cells.

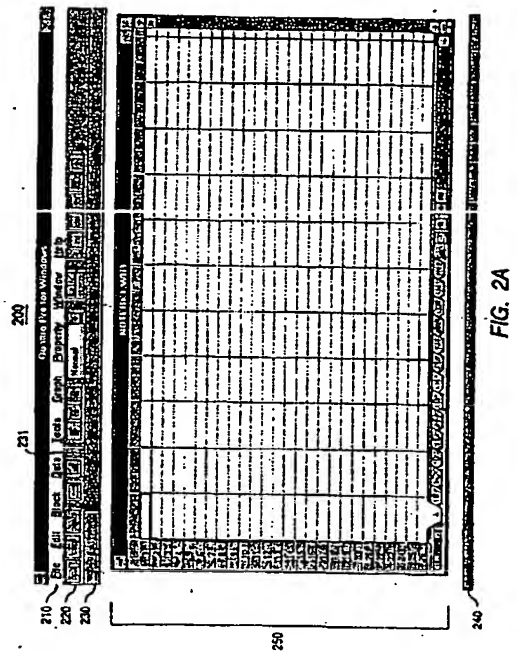


FIG. 2A

EP 0 569 133 A3

BNSDOCID: <EP__0569133A3_I_>

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 93 30 2717

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.5)
X Y	US-A-5 072 412 (HENDERSON ET AL) * column 41, line 44 - column 43, line 6 *	22 1-13,15	G06F15/20 G06F9/44 G06F3/033
Y	CHIP ZEITSCHRIFT FUR MIKROCOMPUTER-TECHNIK no. 3 , March 1991 , WURZBURG DE pages 450 - 454 XP000209136 I. BIRKHOLZ 'Die Datenbank in der Tabellenkalkulation' * page 454, middle column; figure 5 *	1-13,15	
P,X	WO-A-92 08199 (GO CORP.) * page 4, line 20 - line 25 * * page 9, line 30 - page 10, line 7 * * figures 36,37 *	1-22	
A	EP-A-0 428 261 (LOTUS) * column 2, line 7 - column 3, line 6 * * column 3, line 53 - column 4, line 3 *	16-20	
A	IBM TECHNICAL DISCLOSURE BULLETIN. vol. 33, no. 9 , February 1991 , ARMONK US pages 283 - 284 XP000109528 'Hyperpaper' * the whole document *	1,9,22	TECHNICAL FIELDS SEARCHED (Int.Cl.5) G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	14 December 1993	Pottiez, M

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

BNSDOCID: <EP___0569133A3_I_>